

# 深入浅出深度学习

## 原理剖析与Python实践

黃安埠 著



電子工業出版社

Publishing House of Electronics Industry

北京·BEIJING

## 内 容 简 介

本书介绍了深度学习相关的原理与应用，全书共分为三大部分，第一部分主要回顾了深度学习的发展历史，以及 Theano 的使用；第二部分详细讲解了与深度学习相关的基础知识，包括线性代数、概率论、概率图模型、机器学习和最优化算法；在第三部分中，针对若干核心的深度学习模型，如自编码器、受限玻尔兹曼机、递归神经网络和卷积神经网络等进行详细的原理分析与讲解，并针对不同的模型给出相应的具体应用。

本书适合有一定高等数学、机器学习和 Python 编程基础的在校学生、高校研究者或在企业中从事深度学习的工程师使用，书中对模型的原理与难点进行了深入分析，在每一章的最后都提供了详细的参考文献，读者可以对相关的细节进行更深入的研究。最后，理论与实践相结合，本书针对常用的模型分别给出了相应的应用，读者也可以在 Github 中下载和查看本书的代码 (<https://github.com/innovation-cat/DeepLearningBook>)。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。

版权所有，侵权必究。

## 图书在版编目 (CIP) 数据

深入浅出深度学习：原理剖析与 Python 实践 / 黄安埠著. —北京：电子工业出版社，2017.6  
ISBN 978-7-121-31270-0

I. ①深… II. ①黄… III. ①软件工具—程序设计 IV. ①TP311.561

中国版本图书馆 CIP 数据核字(2017)第 070055 号

责任编辑：徐津平

印 刷：

装 订：

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：720×1000 1/16 印张：22.25 字数：401 千字

版 次：2017 年 6 月第 1 版

印 次：2017 年 6 月第 1 次印刷

印 次：3000 册 定价：79.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：(010) 88254888，88258888。

质量投诉请发邮件至 [zltts@phei.com.cn](mailto:zltts@phei.com.cn)，盗版侵权举报请发邮件至 [dbqq@phei.com.cn](mailto:dbqq@phei.com.cn)。

本书咨询联系方式：010-51260888-819，[faq@phei.com.cn](mailto:faq@phei.com.cn)。



# 推荐序 1

介绍深度学习的书籍不少，但是《深入浅出深度学习：原理剖析与 Python 实践》与其他同类书相比，视角明显不同。如果要给本书写个宣传语，或许可以是“深度学习工程师速成培训教材”。本书对读者的知识结构有两点要求：一是学过高等数学，二是熟悉 Python 编程。换言之，各个专业的理工科学生，尤其是学过 Python 编程的，都是此书的目标读者。

本书内容全面，但是取舍明确，有重点地深入，尤其对于技术的重点难点解释得很详细，深入浅出。

本书最大的特色就在于内容取舍的尺度非常明确——着重于原理解释和动手实践的路径，但是并不拘泥于细枝末节。

胸中有经纬，就不会迷失在细节的汪洋大海。本书刚好侧重于对经纬的梳理。此处的经纬，一类是数学基础知识，另一类是深度学习技术。与深度学习相关的数学知识包括线性代数、概率统计等。因为概率图与深度学习结合较多，本书把“概率图”作为单独的一章重点讲述。深度学习技术包括机器学习的传统技术、用于训练神经网络的梯度下降等算法。本书重点讲述了神经网络的基本算法以及几种常用的深度网络架构。

工程师，重在实践。工欲善其事必先利其器，实践深度学习，离不开深度学习工具。本书介绍了 Theano 工具集的基本用法。其实深度学习工具，一通百通，各种工具的区别，类似于北京口音与东北口音的区别。

认真读完此书，读者应该拥有三项能力：一是读得懂深度学习的论文；二是读得

懂深度学习的代码；三是能够自行开发简单的深度学习应用。


总结一下，在深度学习技术异常火爆，深度学习工程师奇缺的当下，如何快速培养深度学习方向的工程师，是一个迫切的问题。此书是难得的好教材。

邓侃博士

---

邓侃,美国卡内基梅隆大学(CMU)计算机学院博士,专攻人工智能和数据挖掘。历任美国甲骨文(Oracle)主任系统架构师,美国泰为手机导航公司(Telenav)北京分公司总经理,百度网页搜索高级总监。2015年至今创建北京大数医达科技有限公司,专注于研发医疗大数据和人工智能医生,出任CTO。





## 推荐序 2

在过去的这十年，深度学习已经席卷了整个科技界和工业界，2016 年谷歌阿尔法狗打败围棋世界冠军李世石，更是使其成为备受瞩目的技术焦点。记得 2010 年 7 月我参与了 Facebook 人脸识别（Face Detection）工作，那时候深度学习还没有普及，这个系统是当时世界上最大的实时人脸识别系统，每天都有几千万张脸被找出来。随着硬件的成熟和数据的指数级增加，深度学习在很多问题上成为人工智能最火和最有效的方法。

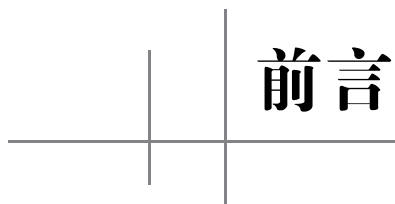
这几年，我身边越来越多的计算机从业者，甚至在校大学生向我咨询，有效了解和学习这个当前人工智能最热门的领域的方法。没错，一方面深度学习很热门，人人都对此充满了好奇和向往，但是同时，它对于初学者往往挑战也不小，因为需要一定的数学基础，同时各种专业术语，如果不用深入浅出的方式讲，很容易让人产生畏难情绪。

所以我推荐黄安埠的这本《深入浅出深度学习：原理剖析与 Python 实践》给所有对深度学习有兴趣的朋友们，本书涵盖了深度学习的理论以及各种常见的深度学习模型，你们会在这本书中找到关于深度学习最实用的知识和信息。对于初学者来说，这是一本非常通俗易懂的入门教材。

同时我也推荐该书给深度学习领域的开发者和数据科学家，因为作者也分享和总结了许多深度学习的最佳实践法，并辅以相当多的实际应用案例加以诠释，是一本值得借鉴参考的好书。

陈尔东

前 Twitter 总监，前 Facebook 经理和早期工程师



# 前言

What magical trick makes us intelligent? The trick is that there is no trick. The power of intelligence stems from our vast diversity, not from any single, perfect principle.

—— Marvin Minsky

智能（Intelligence）这个词的出现最早可以追溯到古希腊时期，当时人们已经开始梦想能创造出一种像人类一样，具有独立思考 and 推理能力的机器，但由于受到当时生产力水平的制约，古人对“智能”的研究更多的是停留在理论探索阶段。到了近代，尤其是具有划时代意义的达特茅斯会议的召开，标志着人工智能开始从理论探索进入到理论与应用相结合的实践阶段。从20世纪50年代开始，人工智能的发展大致经历了三个阶段，分别从最初的逻辑推理，到统计机器学习，再到近年来逐渐占据主流地位的深度学习。

虽然深度学习是一门以神经网络为核心的学科，但人们普遍认为深度学习始于2006年，当时Hinton等人提出基于深度置信网络（DBN）逐层预训练的方法来训练深层模型，并首次提出了深度学习的概念。此后，深度学习开始进入人们的视野，但那时候深度学习更多的是少数顶尖科学家研究的领域，并没有得到大规模的应用和推广。直到2012年，Hinton和他的两个学生Alex Krizhevsky、Illya Sutskever，将卷积神经网络应用到ImageNet竞赛中，并取得了分类错误率15%的成绩，这个成绩比第二名低了近11个百分点，这一历史性的突破，使得人们开始意识到深度学习所拥有的巨大潜力，在这之后，深度学习开始在工业界，尤其是计算机视觉、语音识别和自然语言处理等领域，大规模应用，并且取得了比以往更好的效果。到了2016年，随着AlphaGo的横空出世，它的惊人表现将深度学习的热度推向了顶峰，因此2016年也被很多学者认

为是人工智能元年，事实上，当前人工智能已经影响到人们生活的各个方面，如语音助手、语音搜索、无人驾驶汽车、人脸识别等，为人们的生活带来了极大的方便，人工智能也必将在今后相当长的一段时间内，继续推动着人类的技术发展。

在本书编写的过程中，市面上有关深度学习方面的中文书籍较少，因此作者希望能从理论和应用相结合的角度，对深度学习的相关知识进行较为全面的梳理，本书既可以作为初级读者的入门书籍，也适合中级读者用来加深对理论知识的理解。本书覆盖了线性代数、概率论、数值计算与最优化等基础知识，以及深度学习的两大核心：概率图模型和深度神经网络。具体来说，本书由以下三大部分构成：

第 1 部分是概要，共分为两章。第 1 章主要阐述了深度学习、人工智能相关的背景，深度学习的原理，以及当前流行的深度学习框架对比；第 2 章介绍了深度学习框架Theano的使用，着重对Theano的基础知识和编程范式进行了讲解。

第 2 部分是与深度学习相关的数学和机器学习方面的基础知识，共分为 5 章。第 3 章介绍线性代数基础知识；第 4 章介绍了概率论和数理统计相关的知识；第 5 章介绍概率图模型，包括贝叶斯网络和马尔科夫网络的原理；第 6 章简要回顾机器学习的基础知识，并介绍机器学习模型与深度学习模型之间的联系；第 7 章，深入分析几种常用的机器学习最优化方法，包括具有一阶收敛速度的梯度下降法和共轭梯度法，以及具有二阶收敛速度的牛顿法和拟牛顿法。

第 3 部分介绍了各种常见的深度学习模型，包括一系列的深度学习模型理论及其应用，本部分共分为 6 章。第 8 章介绍全连接前馈神经网络，包括网络结构和激活函数的相关知识；第 9 章将深入分析反向传播算法，以及梯度消失问题。梯度消失也是深度神经网络训练的一大难点，我们将介绍当前有效解决深度网络训练中过拟合和欠拟合的常见技巧，包括Batch Normalization、残差网络、Dropout等；第 10 章介绍本书的第一种无监督网络模型：自编码器及其变种模型；第 11 章介绍一种深度概率图模型——受限玻尔兹曼机，与自编码器一样，受限玻尔兹曼机也是一种常见的无监督网络模型，最后介绍如何将受限玻尔兹曼机应用于个性化推荐领域中；第 12 章，将介绍一种应用非常广泛的网络结构——递归神经网络，深入分析递归网络的结构及其变形网络，如LSTM、GRU等，并以语言模型为例，介绍递归神经网络在自然语言处理中的应用；第 13 章介绍另一种常见的模型结构：卷积神经网络，包括卷积网络的卷积层和池化层结构设计，以及其在文本分类中的应用。

关于本书的源代码，读者也可以从Github上（<https://github.com/innovation->

cat/DeepLearningBook) 下载查看。深度学习近年来处于高速发展的阶段, 很多更先进的理论和算法正被不断提出, 因此本书无法覆盖所有的模型与算法, 加之作者水平和精力所限, 书中难免有错漏之处, 承蒙各位读者不吝告知, 如对本书有任何疑问或建议, 读者可以通过邮箱 [huanganbu@gmail.com](mailto:huanganbu@gmail.com) 给我反馈。

在本书的撰写过程中, 得到了很多行业专家和好友的支持, 在此, 特别感谢香港科技大学计算机系主任杨强教授、原百度网页搜索高级总监邓侃博士、原Twitter工程总监陈尔东先生, 感谢他们在百忙之中抽时间审阅我的书稿, 提出了很多宝贵的意见, 并为我写下推荐序。

在本书的撰写过程中, 还得到了电子工业出版社刘皎编辑和汪达文编辑的极大帮助, 在此表示衷心的感谢; 感谢我在腾讯公司的上级李深远先生对我工作的支持, 也感谢其他各位关心我工作的朋友和同事。

最后, 非常感谢我的家人对我工作的理解和支持, 他们在我写作的过程中给予了很大的照顾和鼓励, 也是促使我能完成本书写作的最大动力。

黄安埠

2017年3月于深圳

---

轻松注册成为博文视点社区用户 ([www.broadview.com.cn](http://www.broadview.com.cn)), 扫码直达本书页面。

- **下载资源:** 本书如提供示例代码及资源文件, 均可在 [下载资源](#) 处下载。
- **提交勘误:** 您对书中内容的修改意见可在 [提交勘误](#) 处提交, 若被采纳, 将获赠博文视点社区积分 (在您购买电子书时, 积分可用来抵扣相应金额)。
- **交流互动:** 在页面下方 [读者评论](#) 处留下您的疑问或观点, 与我们和其他读者一同学习交流。

页面入口: <http://www.broadview.com.cn/31270>





# 目录

第 1 部分 概要 .....	1
1 绪论 .....	2
1.1 人工智能、机器学习与深度学习的关系 .....	3
1.1.1 人工智能——机器推理 .....	4
1.1.2 机器学习——数据驱动的科学 .....	5
1.1.3 深度学习——大脑的仿真 .....	8
1.2 深度学习的发展历程 .....	8
1.3 深度学习技术概述 .....	10
1.3.1 从低层到高层的特征抽象 .....	11
1.3.2 让网络变得更深 .....	13
1.3.3 自动特征提取 .....	14
1.4 深度学习框架 .....	15
2 Theano 基础 .....	19
2.1 符号变量 .....	20
2.2 符号计算的抽象——符号计算图模型 .....	23
2.3 函数 .....	26
2.3.1 函数的定义 .....	26
2.3.2 Logistic 回归 .....	27
2.3.3 函数的复制 .....	29
2.4 条件表达式 .....	31

2.5	循环.....	32
2.6	共享变量.....	39
2.7	配置.....	39
2.7.1	通过THEANO_FLAGS配置 .....	40
2.7.2	通过.theanorc文件配置 .....	41
2.8	常用的Debug技巧.....	42
2.9	小结.....	43
<b>第2部分 数学与机器学习基础篇 .....</b>		<b>45</b>
3	线性代数基础.....	46
3.1	标量、向量、矩阵和张量.....	46
3.2	矩阵初等变换.....	47
3.3	线性相关与向量空间.....	48
3.4	范数.....	49
3.4.1	向量范数.....	49
3.4.2	矩阵范数.....	53
3.5	特殊的矩阵与向量 .....	56
3.6	特征值分解.....	57
3.7	奇异值分解.....	58
3.8	迹运算.....	60
3.9	样例：主成分分析 .....	61
4	概率统计基础.....	64
4.1	样本空间与随机变量.....	65
4.2	概率分布与分布函数.....	65
4.3	一维随机变量 .....	66
4.3.1	离散型随机变量和分布律 .....	66
4.3.2	连续型随机变量和概率密度函数 .....	67
4.4	多维随机变量 .....	68
4.4.1	离散型二维随机变量和联合分布律 .....	69
4.4.2	连续型二维随机变量和联合密度函数 .....	69
4.5	边缘分布 .....	70
4.6	条件分布与链式法则.....	71

4.6.1	条件概率.....	71
4.6.2	链式法则.....	73
4.7	多维随机变量的独立性分析.....	73
4.7.1	边缘独立.....	74
4.7.2	条件独立.....	74
4.8	数学期望、方差、协方差.....	75
4.8.1	数学期望.....	75
4.8.2	方差.....	76
4.8.3	协方差.....	76
4.8.4	协方差矩阵.....	78
4.9	信息论基础.....	81
4.9.1	信息熵.....	81
4.9.2	条件熵.....	83
4.9.3	互信息.....	84
4.9.4	相对熵与交叉熵.....	84
5	概率图模型.....	87
5.1	生成模型与判别模型.....	89
5.2	图论基础.....	90
5.2.1	图的结构.....	90
5.2.2	子图.....	91
5.2.3	路径、迹、环与拓扑排序.....	92
5.3	贝叶斯网络.....	95
5.3.1	因子分解.....	96
5.3.2	局部马尔科夫独立性断言.....	99
5.3.3	I-Map与因子分解.....	100
5.3.4	有效迹.....	103
5.3.5	D-分离与全局马尔科夫独立性.....	108
5.4	马尔科夫网络.....	108
5.4.1	势函数因子与参数化表示.....	109
5.4.2	马尔科夫独立性.....	111
5.5	变量消除.....	114
5.6	信念传播.....	116
5.6.1	聚类图.....	116

5.6.2	团树 .....	120
5.6.3	由变量消除构建团树 .....	123
5.7	MCMC采样原理 .....	126
5.7.1	随机采样 .....	127
5.7.2	随机过程与马尔科夫链 .....	128
5.7.3	MCMC采样 .....	132
5.7.4	Gibbs采样 .....	134
5.8	参数学习 .....	137
5.8.1	最大似然估计 .....	137
5.8.2	期望最大化算法 .....	138
5.9	小结 .....	140
6	机器学习基础 .....	142
6.1	线性模型 .....	143
6.1.1	线性回归 .....	143
6.1.2	Logistic回归 .....	148
6.1.3	广义的线性模型 .....	150
6.2	支持向量机 .....	151
6.2.1	最优间隔分类器 .....	152
6.2.2	对偶问题 .....	155
6.2.3	核函数 .....	156
6.3	朴素贝叶斯 .....	160
6.4	树模型 .....	162
6.4.1	特征选择 .....	163
6.4.2	剪枝策略 .....	165
6.5	聚类 .....	166
6.5.1	距离度量 .....	167
6.5.2	层次聚类 .....	168
6.5.3	K-means聚类 .....	171
6.5.4	谱聚类 .....	172
7	数值计算与最优化 .....	177
7.1	无约束极小值的最优化条件 .....	177
7.2	梯度下降 .....	179



7.2.1	传统更新策略.....	181
7.2.2	动量更新策略.....	183
7.2.3	改进的动量更新策略.....	184
7.2.4	自适应梯度策略.....	187
7.3	共轭梯度.....	188
7.4	牛顿法.....	192
7.5	拟牛顿法.....	194
7.5.1	拟牛顿条件.....	194
7.5.2	DFP算法.....	195
7.5.3	BFGS算法.....	196
7.5.4	L-BFGS算法.....	197
7.6	约束最优化条件.....	200
<b>第3部分 理论与应用篇.....</b>		<b>205</b>
8	<b>前馈神经网络.....</b>	<b>206</b>
8.1	生物神经元结构.....	207
8.2	人工神经元结构.....	208
8.3	单层感知机.....	209
8.4	多层感知机.....	212
8.5	激活函数.....	217
8.5.1	激活函数的作用.....	217
8.5.2	常用的激活函数.....	219
9	<b>反向传播与梯度消失.....</b>	<b>225</b>
9.1	经验风险最小化.....	227
9.2	梯度计算.....	228
9.2.1	输出层梯度.....	228
9.2.2	隐藏层梯度.....	230
9.2.3	参数梯度.....	234
9.3	反向传播.....	235
9.4	深度学习训练的难点.....	237
9.4.1	欠拟合——梯度消失.....	237
9.4.2	过拟合.....	240

10	自编码器及其相关模型 .....	243
10.1	自编码器 .....	243
10.2	降噪自编码器 .....	245
10.3	栈式自编码器 .....	247
10.4	稀疏编码器 .....	250
10.5	应用：cifar10图像分类 .....	254
11	玻尔兹曼机及其相关模型 .....	258
11.1	玻尔兹曼机 .....	258
11.2	能量模型 .....	261
11.2.1	能量函数 .....	261
11.2.2	从能量函数到势函数 .....	262
11.2.3	从势函数到概率分布 .....	263
11.3	推断 .....	264
11.3.1	边缘分布 .....	265
11.3.2	条件分布 .....	267
11.4	学习 .....	270
11.4.1	最大似然估计 .....	271
11.4.2	对比散度 .....	274
11.5	应用：个性化推荐 .....	276
11.5.1	个性化推荐概述 .....	276
11.5.2	个性化推荐架构与算法 .....	279
11.5.3	RBM与协同过滤 .....	285
12	递归神经网络 .....	291
12.1	Elman递归神经网络 .....	292
12.2	时间反向传播 .....	295
12.3	长短时记忆网络 .....	299
12.4	结构递归神经网络 .....	302
12.5	应用：语言模型 .....	308
12.5.1	N元统计模型 .....	308
12.5.2	基于 LSTM 构建语言模型 .....	312

13	卷积神经网络.....	318
13.1	卷积运算.....	319
13.2	网络结构.....	320
13.3	卷积层.....	324
13.4	池化层.....	329
13.5	应用：文本分类.....	333



---

# 第 1 部分

## 概 要

---

# 1

## 绪论

远古希腊时期，科学家就有一个梦想，梦想创建一种能像人类一样，具有独立思考 and 推理能力的机器。从那时候开始，数理逻辑和认知逻辑等分支在数千年的积累和发展过程中，总结出大量有规律性的定理法则，这些定理法则为科学研究提供了方法论层面的指导。

直到20世纪中期，第一台可编程计算机的出现，才使得人类实现人工智能（Artificial Intelligence，简称 AI）的梦想成为可能。早期科学家们总结的逻辑推理规则，通过机器指令的形式输入到计算机，使得计算机具有了初级的推理能力。事实上，从20世纪50年代到70年代，科学家们也普遍认为人工智能应该解决的是，那些对人类来说非常困难，但对计算机来说相对简单直接的任务，比如数据量和运算量都非常巨大的复杂的数学问题，对人类来说，这类问题不可能由人工手动来完成，但对计算机来说却异常简单，只需要输入相应的指令和数学规则，凭借计算机强大的运算能力，问题就能够轻松解决。但后来人们发现，人工智能真正的挑战，应该是那些对人类来说非常直观，但对计算机来说却难以用指令规则来描述的任务，比如人类能很容易识别出眼前的动物是一只猫还是一只狗，但对计算机来说，这个任务在当时却异常困难。

以机器学习（Machine Learning，简称 ML）为代表的第二代智能算法的出现，使得这些任务得到了有效的解决，从而带动了人工智能的新一轮发展，在这段时间里，机器学习技术在多个领域，包括自然语言处理、计算机视觉等，都取得了重大的突破。然而，到了21世纪初，机器学习的发展同样出现了瓶颈，这主要是由于机器学习的大

多数算法都是一种浅层学习，无法有效学习到数据的深层次特征，使得人工智能始终没有办法取得进一步的突破。随着深度学习（Deep Learning，简称 DL）的出现，这种状况才得以打破，和以往不同的是，深度学习的出现，不仅可以让弱人工智能的任务，在性能上较传统的机器学习算法有了进一步的提升，更让人们看到实现通用人工智能的希望。

本章将对深度学习的历史和技术进行简单的介绍，然后介绍当前常用的深度学习开源框架，我们将比较这些框架在不同维度上的表现。

## 1.1 人工智能、机器学习与深度学习的关系

人工智能、机器学习和深度学习是当前机器智能领域最热门的三个词汇，很多人甚至将三者看成是一种等价的关系，例如在2016年3月，当Google Deepmind的AlphaGo击败了韩国围棋大师李世石九段后，媒体在报道时就混杂使用了人工智能、机器学习和深度学习等多种术语。事实上，这三者之间既有一定的联系，但也有明显的区别。要正确理解深度学习的概念，首先应该了解人工智能、机器学习与深度学习这三者之间的关系。

要理解三者之间的关系，可以通过同心圆来可视化表示三者的关系。最外面的圆环代表人工智能，里面一层表示机器学习，人工智能和深度学习处于中心位置，也可以简单理解为机器学习是人工智能的一个分支，而深度学习则是一种特殊的机器学习实现方法，如图1.1所示。

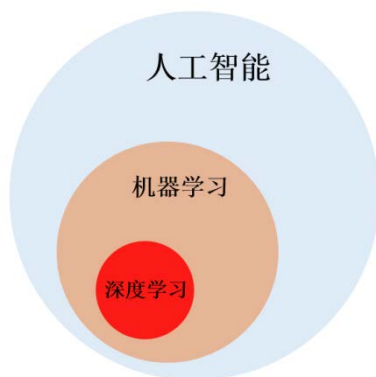


图1.1 人工智能、机器学习、深度学习三者的关系

1.1.1 人工智能——机器推理

1956年夏天，包括约翰·麦卡锡、马文·明斯基、罗切斯特和香农在内的10位顶级科学家，在达特茅斯学院召开了“达特茅斯夏季人工智能研讨会”，这次会议被广泛认为是现代人工智能研究的诞生之日。图1.2是1956年达特茅斯会议的参与者。

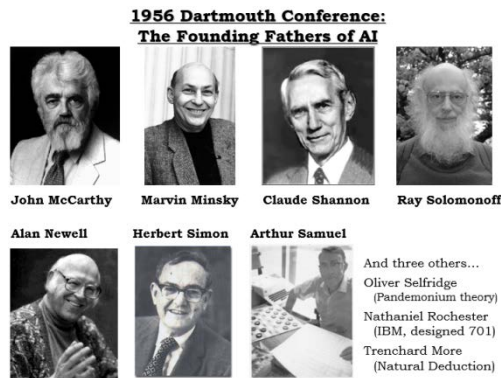


图1.2 1956年达特茅斯会议的参与者

会议的议题覆盖了人工智能的各个领域，包括：神经网络、自然语言处理、机器智能等。从这次会议之后，科学家们一直梦想实现由新兴计算机构建的具有人类智力特征的复杂机器，这就是所谓的**通用人工智能**（General Artificial Intelligence）或**强人工智能**，即让机器拥有人类的所有感觉、所有理智，像人类一样思考。虽然这种机器到目前为止仍然没有成为现实，但这并没有阻止人们对通用人工智能的探索和想象，事实上，我们已经在很多好莱坞电影中看到过这种机器，比如《星球大战》中的C-3PO，或者“终结者”系列中成为人类敌人的机器——人型机械人T-800。

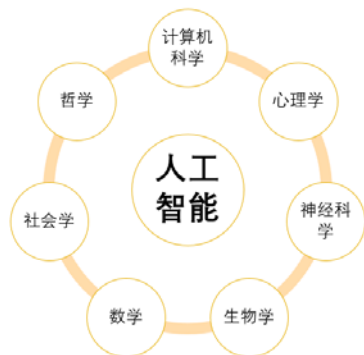


图1.3 与人工智能相关的学科领域，人工智能是一个跨学科的领域，覆盖了计算机科学、哲学、心理学、神经科学、数学等领域知识



要实现真正意义上的通用人工智能，或许还有很漫长的路要走，但在一些特定的领域，或者一些特殊的任务，我们希望能让机器处理得同人类一样好，甚至比人类更加出色，比如图像识别、人脸识别、计算机视觉等领域。这些领域或任务也被称为**狭义的人工智能**（Narrow Artificial Intelligence）或**弱人工智能**。

当前在狭义的人工智能领域，人类已经取得了很大的突破，某些工作机器甚至比人类做得要好。例如，在2016年10月28日，微软的雷蒙德研究院开发出一种新算法，使计算机对指定主题对话的语音识别率提升至94.1%，与人类水平相当；对亲戚朋友日常对话的识别率高达88.9%，甚至比人类略胜一筹。而取得这些突破性进展的背后，依靠的是机器强大的学习能力，这就是接下来将要讨论的第二个圆环——机器学习。

### 1.1.2 机器学习——数据驱动的科学

机器学习，也被称为统计机器学习，是人工智能领域的一个分支，其基本思想是基于数据构建统计模型，并利用模型对数据进行分析和预测的一门学科。

传统上，如果想让计算机工作，我们会编写一段指令，然后让计算机遵照这个指令一步一步执行下去。而机器学习则是采用另一种解决问题的思路，机器学习解决问题的方式不是通过输入指令逻辑，而是通过输入的数据，也就是说，机器学习是一种让计算机利用数据而不是指令来进行各种工作的方法。

机器学习最基本的做法是使用算法来解析数据，从数据中学习规律，并掌握这种规律，然后对真实世界中的事件做出决策或预测。与传统的为解决特定任务、硬编码的软件程序不同，机器学习的核心是使用大量的数据来训练，通过各种算法从数据中学习如何完成任务。机器学习直接来源于早期的人工智能领域，在模式识别和机器学习理论的研究中逐渐发展，并最终形成一门新的学科。与人工智能类似，机器学习也是一个跨学科领域，涉及多个基础学科，包括统计学、线性代数和数值计算等。

前面提到，机器学习是基于训练数据构建统计模型，从而使计算机具有对新数据进行预测和分析的能力，机器学习方法按其实现的目标不同，可以分为：监督学习、无监督学习和强化学习。

**监督学习（Supervised Learning）**：监督学习使用带有标签的训练数据集进行训练，输入的训练数据由物体的特征向量（输入）和物体的标签（输出）两部分构成，其中，若输出的标签是一个连续的值，则称为回归监督学习；若输出标签是一个离散的值，则称为分类监督学习。

监督学习涉及两个方面的工作：首先，根据提供的训练数据，选择一种合适的模型进行训练，直至模型的训练收敛。常见的监督学习模型包括：Logistic回归、决策树、SVM（Support Vector Machines，支持向量机）、KNN、朴素贝叶斯等。图1.4展示的是一个水果分类的例子，每一个样本数据的输入是由物体的特征构成的特征向量，如物体的颜色、大小、形状等，输出的是物体的类别，如苹果、葡萄、香蕉等。

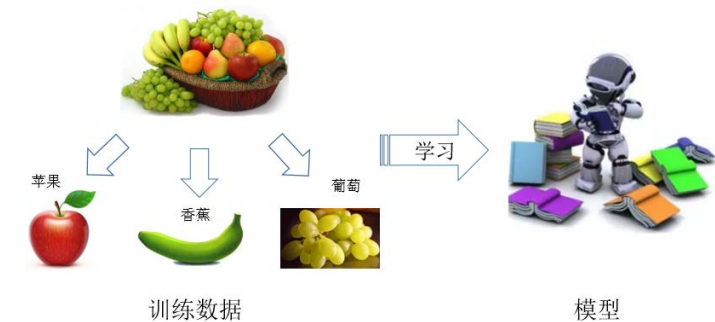


图1.4 监督学习模型训练，算法利用训练数据提供的特征信息，如颜色、大小、形状等，构建概率模型 $p(y|x)$ 或非概率模型 $y = f(x)$

其次，当模型训练完毕，就可以把新的输入数据代入模型，模型将根据新数据的特征信息，找出最符合这种特征的输出结果，其过程如图1.5所示。



图1.5 模型预测

**无监督学习（Unsupervised learning）：**无监督学习的训练样本数据没有任何的标签和输出，其目的是对原始数据结构进行深入分析，找出数据间存在的规律与关系。典型的无监督学习任务包括：聚类、降维、特征提取等。

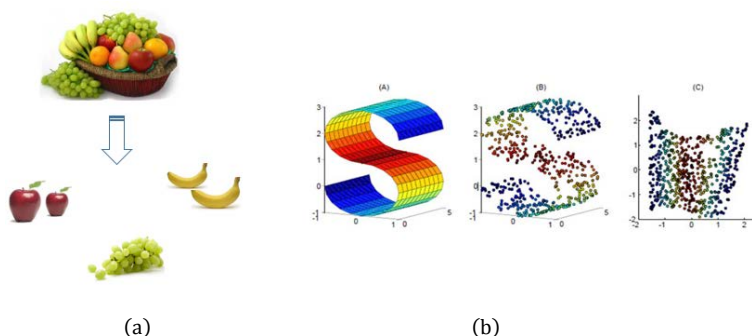


图1.6 两种常见的无监督学习, (a)数据聚类, (b)数据降维

虽然监督学习的准确率更高,但在现实生活中,我们获取的大量数据一般是没有标签数据的,因此,我们不得不诉诸于无监督学习,但传统的无监督学习方法在特征提取上并不令人满意,而深度学习则被证明具有强大的无监督学习能力,特别是在计算机视觉领域,运用深度学习技术所达到的效果更是要远优于传统的机器学习。

**强化学习 (reinforcement learning):** 强化学习也称为增强学习,强调如何基于环境而行动,以取得最大化的预期利益。其灵感来源于心理学中的行为主义理论,即有机体如何在环境给予的奖励或惩罚的刺激下,逐步形成对刺激的预期,产生能获得最大利益的习惯性行为<sup>[1,2]</sup>。

强化学习与前面的监督学习、无监督学习之间的区别在于,它并不需要出现正确的输入输出对,也不需要精确校正次优化的行为。强化学习更加专注于在线规划,需要在探索未知的领域和遵从现有知识之间找到平衡,它的学习过程是一个从实际环境中不断学习积累,不断进化的过程。因此,强化学习更接近生物学习的本质,也是有望让机器获得通用智能的一项技术。

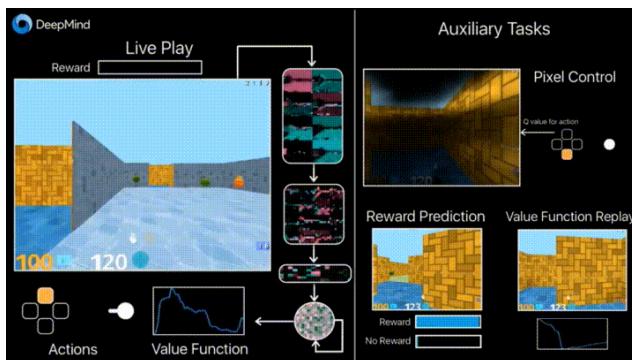


图1.7 DeepMind利用强化学习技术在迷宫游戏中执行搜索任务 (图片摘自网络)

### 1.1.3 深度学习——大脑的仿真

过去，深度学习是作为机器学习的一个算法而存在，被称为人工神经网络，由于受到算法理论、数据和硬件的制约，多年以来，神经网络都是单层或浅层的网络结构，并且随着其他更有效率的浅层算法，如SVM、Logistic回归的提出，神经网络在效果和性能上都没有任何优势，因此，神经网络也逐渐淡出了人们的视野。但随着大数据的发展，以及大规模硬件加速设备的出现，特别是GPU运算性能的不断提升，使得神经网络重新受到人们的重视。

除了大数据和高性能计算平台的推动，真正让人们感受到深度学习强大威力的，是深度学习在技术上的一系列创新和突破，包括从低层到高层的特征抽象、特征自动提取、layer-wise解决梯度消失、深度残差技术、生成对抗网络等，我们将在本书的其他章节中详细介绍其中的一部分技术。

正是由于深度学习近十年来在理论上的不断创新，以及在商业应用中取得的突破性进展，使得人工智能迅速转入到当前的深度学习时代，深度学习也因此被MIT技术评论列为2013年十大突破性技术之首<sup>[3]</sup>。

## 1.2 深度学习的发展历程

深度学习，或其前身人工神经网络，其实已经有超过60年的历史，在这段时期里，神经网络的发展也经历了多次的起伏。

神经网络的第一次高潮出现在20世纪50年代。1957年，计算科学家Rosenblatt提出了感知器的概念，即由输入层和输出层构成的无隐藏层神经网络，这也是神经网络第一次出现在大众的视野，Rosenblatt还现场演示了如何利用感知器来学习识别简单图像的过程，这在当时的社会上引起了极大的轰动，人们第一次看到机器是如何通过学习来获得智能，至此，许多学者和科研机构纷纷投入到神经网络的研究中。美国军方也大力资助了神经网络的研究，这持续到1969年才结束。

造成神经网络从高潮走向低潮的原因在于，单层的感知机无法解决非线性数据的分类，它甚至无法解决简单的“异或”问题，虽然后来有学者提出多层的神经网络能够解决这类非线性问题，但是却没有提出多层神经网络的有效训练方法。

神经网络的第二次高潮出现在20世纪80年代，以反向传播算法的提出为标志。1986年，Rumelhar和Hinton等人提出了反向传播(Backpropagation，简称BP)算法<sup>[5]</sup>，

解决了两层乃至多层的神经网络训练问题，过去三十年一直无法解决的非线性分类问题被彻底攻克，从而带动了业界对神经网络研究的第二次浪潮。

第二次高潮一直持续到20世纪90年代中，人们开始发现利用BP算法求解多层神经网络存在很多的制约：首先，随着神经元节点的增多，训练的时间也变得越长。第二个制约是神经网络的优化函数是一个非凸优化问题，往往容易造成局部最优解。而更严重的是第三个问题，从理论上来说，网络层数越多，神经网络的学习能力越强，但人们发现，随着网络层数的增多，网络的学习能力并没有随之提高，这在后来被证明了主要是由于BP算法导致的梯度消失。另一方面，在这段时期，由Vapnik等人发明的SVM算法诞生，很快就在若干个方面体现出了对比神经网络的优势，如无须调参、高效、全局最优解等。基于以上种种理由，SVM迅速打败了神经网络算法，成为了机器学习的主流。而神经网络也因此进入了第二次的低潮时期。

2006年，Geoffery Hinton在*Science*杂志上发表了论文，首次提出了“深度信念网络”的概念<sup>[6]</sup>。传统的训练方式采用随机初始化的方式来初始化权重参数，与传统的训练方式不同，深度信念网络有一个预训练（pre-training）的过程，这样可以方便神经网络中的权值找到一个接近最优解的初始值，再使用“微调（fine-tuning）”技术来对整个网络进行优化训练。这两个技术的运用大幅度减少了训练多层神经网络的时间，并且有效地缓解BP算法导致的梯度消失问题，它给多层神经网络相关的学习方法赋予了一个新名词——**深度学习**，这也标志着神经网络第三次高潮时期的到来。

而真正使深度学习开始受到世人瞩目的，是在2012年的ImageNet竞赛上，Hinton与他的学生在用多层的卷积神经网络成功地对包含一千类别的一百万张图片进行了训练，取得了分类错误率15%的好成绩，这个成绩比第二名低了近11个百分点，此后，深度学习方法开始在工业上和学术上进入了爆发式发展时期。

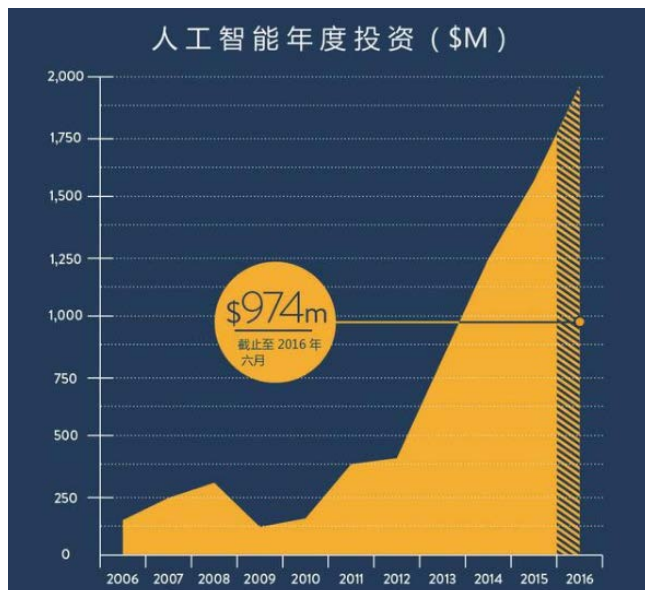


图1.8 在2016年的前6个月时间里，人工智能已获得9.74亿美元的投资。2026年的总投资额必定会超过2015年的总投资额，并且CBInsights指出，200家人工智能公司已获得了近15亿美元的融资（图片与数据来源于Venture Scanner 2016）

当然，深度学习的兴起离不开大数据和高性能计算平台的推动，它们分别被称为深度学习的“燃料”和“引擎”。众所周知，深度学习的成功需要依靠大量的训练数据来进行学习，大数据是深度学习的基础。例如，AlphaGo在与李世石对战时，就已经学习了人类的15万盘棋谱，并且在此基础上，还学习了3000万盘自我对弈的棋谱才达到当前的水平。另一方面，要对大量的数据进行学习和训练，效率问题就成为制约深度学习进一步发展的一大难题，但随着高性能计算平台的不断发展，使得当前数据的处理速度相比十年前有了很大的提升，尤其是GPU技术的发展，GPU拥有出色的浮点数计算性能、超高的并行度，和优化的矩阵运算能力，特别适合于深度学习两大关键步骤：分类和卷积，并且在相同的精度下，相对传统CPU处理数据的方式，拥有更快的处理速度、更少的服务器投入和更低的功耗。

### 1.3 深度学习技术概述

计算机技术在过去几十年取得了长足的进步，但长期以来，人工智能一直处于弱人工智能阶段，直到深度学习的出现，才让人们看到实现强人工智能的曙光。下面我们来简要概括深度学习在三个技术层面上带来的变化。

### 1.3.1 从低层到高层的特征抽象

1.2节提到了深度学习的发展经历的三个阶段,而这三个阶段都以神经网络为核心,它是模拟大脑工作的一种智能算法,让我们首先来了解大脑是如何工作的。

诺贝尔医学奖获得者David Hubel发现了人的视觉系统是以一种分级的方式对外部信息进行处理。如图1.9所示,从视网膜出发,经过V1区提取物体的边缘特征,到V2区的基本形状或目标的局部,再到高层的整个目标(如判定为一张人脸),以及到更高层的PFC(前额叶皮层)进行分类判断等。我们可以发现,大脑对物体输入信号的处理是一种分层的机制,也就是说高层的特征是低层特征的组合,从低层到高层的特征表达越来越具体化和概念化,同时也越来越能表现语义或者意图。

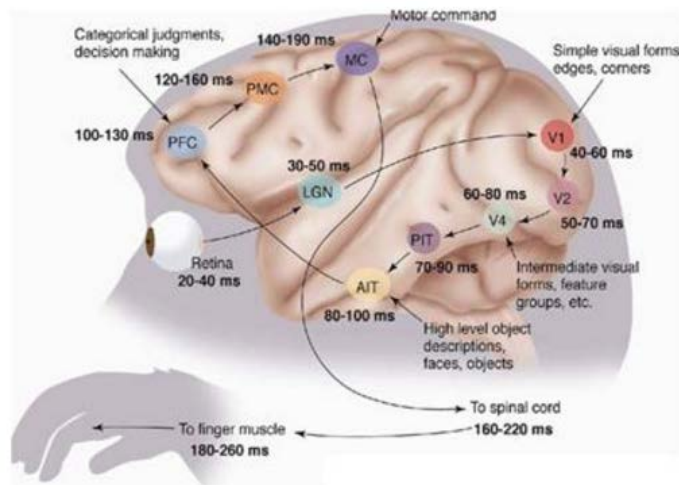


图1.9 人脑的视觉处理系统(图片来源于网络)

视觉系统分层处理机制的发现促进了人们对于神经系统的进一步思考。大脑的工作过程是一个不断迭代、不断抽象概念化的过程,如图1.10所示。首先从原始信号(像素)摄入开始(第一步),接着做初步处理,识别出物体的边缘(第二步),将边缘抽象(大脑判定眼前物体的形状,比如矩形、椭圆形等),然后进一步抽象(大脑进一步形成更复杂的轮廓),最后识别出眼前的物体属于什么类别。这个过程其实和我们的常识是相吻合的,因为复杂的图形,往往就是由一些基本结构组合而成的。同时我们还可以看出:大脑是一个深度架构,认知过程也是深度的。



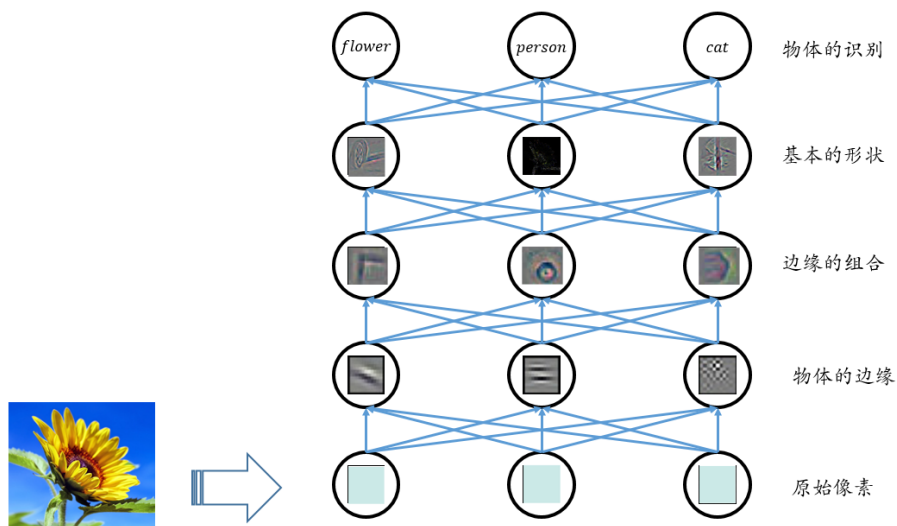


图1.10 深度学习模型处理数据的流程，图片从底层像素点开始，通过不断地组合和抽象，最终到达高层的物体识别

深度学习恰恰就是通过组合低层特征形成更加抽象的高层特征。例如，在计算机视觉领域，深度学习算法从原始图像的像素数据出发，通过不同的卷积核处理，如拉普拉斯滤波器均值滤波器等（可参考第13章卷积神经网络的相关知识），去学习得到一个低层次表达，然后在这些低层次表达的基础上，通过线性或者非线性组合，来获得一个高层次的表达。此外，不仅图像存在这个规律，声音也是类似的。研究人员从声音库中通过算法自动发现了基础的声音结构，而所有的音频数据都可以由这些基本结构通过组合的方式得到，如图1.11所示。

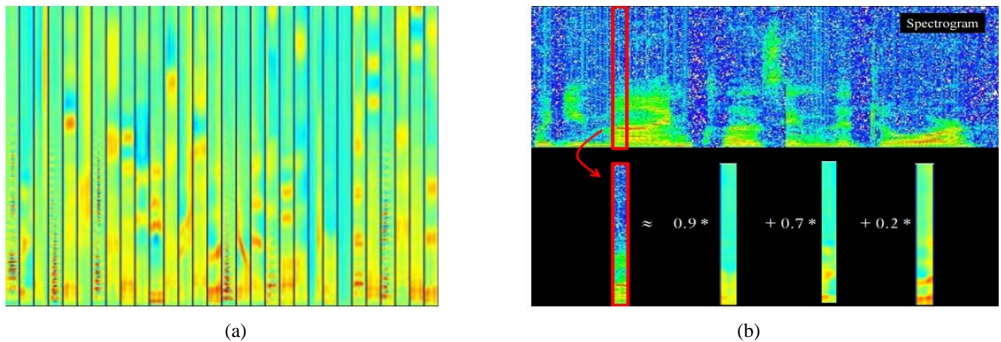


图1.11 任意一段音频数据，都可以通过基础的声音结构组合而成( 图片摘自Andrew Ng的文献[7] )



### 1.3.2 让网络变得更深

深度学习是机器学习研究中的一个分支学科，深度学习的目的在于建立可以模拟人脑进行分析学习的模型，它模仿人脑的机制来解释数据，例如，图像、声音和文本。深度学习之所以被称为“深度”，是因为之前的机器学习方法大都是浅层学习。深度学习可以简单理解为传统神经网络的发展。大约二三十年前，神经网络曾经是机器学习领域特别热门的一个方向，这种基于统计的机器学习方法比起过去基于人工规则的专家系统，在很多方面显示出优越性。如图1.12所示，深度学习与传统的神经网络之间有相同的地方，采用了与神经网络相似的分层结构：系统是一个包括输入层、隐藏层（可单层、可多层）、输出层的多层网络，只有相邻层节点（单元）之间有连接，而同一层以及不相邻的层节点之间相互不连接。这种分层结构，比较接近人类大脑的结构。

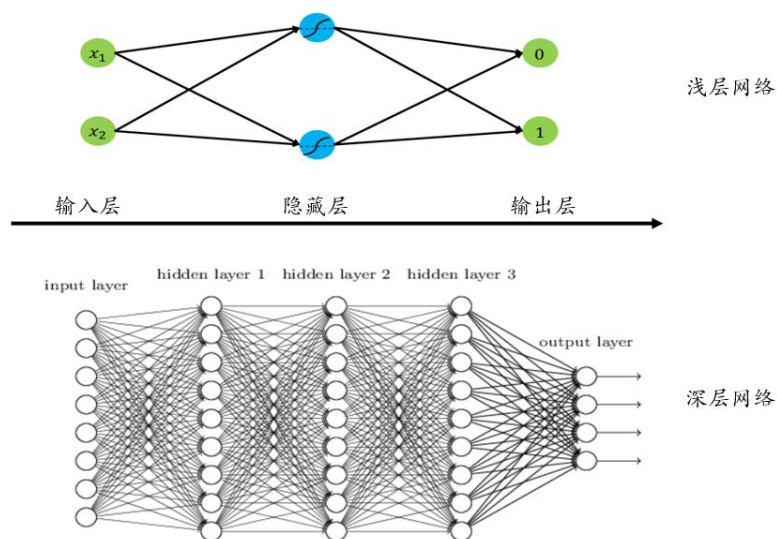


图1.12 传统的神经网络与深度神经网络

但是后来，因为理论分析的难度，加上训练方法需要很多经验和技巧，以及巨大的计算量和优化求解难度，神经网络慢慢淡出了科研领域的主流方向。采用BP算法求解的神经网络在层次深的情況下性能变得很不理想，容易出现所谓的梯度消失现象，且这种情况随着网络层数的增加而更加严重，所以当时的神经网络大多采用浅层结构。2006年，Hinton等人提出了预训练与微调相结合的技术来对整个网络进行优化训练。这两个技术的运用大幅度减少了训练多层神经网络的时间，并且有效缓解BP算法导致

的梯度消失问题。此后，各种优化技术不断出现，如单侧抑制的激活函数ReLU取代传统的sigmoid激活函数，使得梯度消失的问题得到进一步的缓解。最近，一种称为深度残差的技术被应用到神经网络的训练中，使得网络层数达到了百层以上<sup>[8]</sup>。

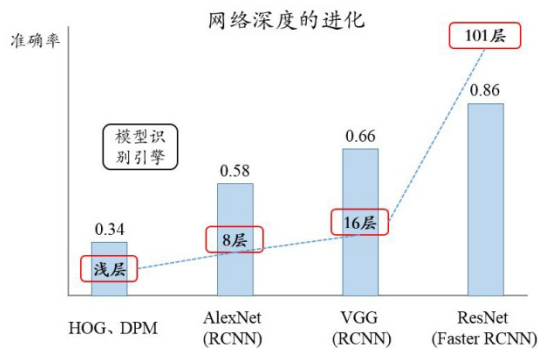


图1.13 深度学习中的网络层数越来越深（图片摘自文献[8]）

### 1.3.3 自动特征提取

深度学习带来的第三个技术性改革是它具有强大的自动提取特征的能力。浅层结构算法有很多局限性，在有限样本和计算单元情况下对复杂函数的表示能力有限，针对复杂分类问题其泛化能力受到一定的制约。更重要的是，浅层模型有一个特点，就是需要依靠人工来抽取样本的特征。然而，手工选取特征是一件非常费力的事情，能不能选取好的特征很大程度上靠经验和运气。既然手工选取特征不太好，那么能不能自动地学习一些特征呢？

深度学习给出了肯定的答案，深度学习框架将特征提取和分类器结合到一个框架中，自动从海量大数据中去学习特征，在使用中减少了手工设计特征的巨大工作量。相比前两次人工智能高潮中的神经网络模型，深度学习带来的变化不仅仅是层数上简单粗暴的堆叠，更重要的是端到端（end-to-end）的表示学习（representation learning）思想。深度学习时代，输入数据直接变成了欲处理对象的最初形态，如初始图像、初始语音等。这类数据表示与高层概念表示之间往往存在较大鸿沟，而端到端的无缝学习过程将表示学习和分类器训练整体耦合为一个系统，可以较理想地弥合这一鸿沟。

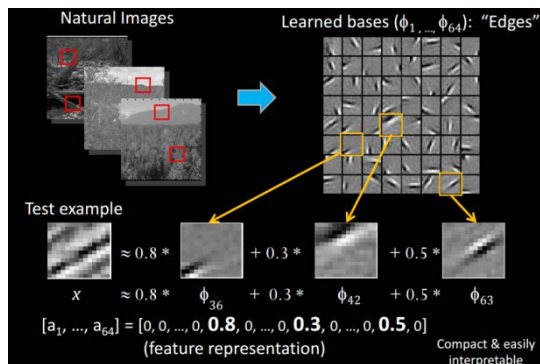


图1.14 利用深度学习自动提取关键特征（图片摘自 Andrew Ng 的文献[7]）

## 1.4 深度学习框架

要深入理解深度学习，关键要多实践，选择一款合适的深度学习框架是进行深度学习的第一步。本节将介绍4种在工业界和学术界都被广泛使用的深度学习开源框架，并从不同的角度来分析和比较它们的优缺点。首先来对这4种深度学习开源框架进行简单的介绍。



**Theano**：是遵循BSD协议的一个开源项目，在2008年，由Yoshua Bengio领导的加拿大蒙特利尔理工学院LISA实验室开发。相比于其他3个框架，Theano的学术气息更浓厚，在学术界和各种数据挖掘竞赛中非常流行，很多先进的算法理论也是首先通过Theano来得到验证。

**Caffe**：全称Convolutional Architecture for Fast Feature Embedding，是一个由C++编写的深度学习框架，最开始是由UC Berkeley的贾扬清博士创建的一个开源项目，目前由伯克利的视觉与学习中心（Berkeley Vision and Learning Center）维护。

**Torch**：诞生时间已经有超过十年之久，但是真正起势得益于2016年Facebook开源了大量Torch的深度学习模块和扩展。Torch的一个特殊之处在于，Torch是一个由Lua语言开发的深度学习框架。Torch目前在Facebook和Twitter内部被广泛使用。

**Tensorflow**：是Google公司在2015年11月开源的新一代机器学习系统，虽然相比

其他开源工具，Tensorflow的发布时间要晚很多，但得益于Google的强大技术支持和不遗余力的宣传，Tensorflow在很短的时间内就吸引了大量的用户，也是2016年Github上最受欢迎的深度学习项目。

如何选择适合自己的深度学习框架？这需要根据读者的实际情况来决定，本节的目的不是对比哪一种框架更有优势，而是从几个不同的维度来比较这4个深度学习工具库之间的差异，以及各自设计的特点。需要强调的是，这些开源框架仍然在不断的发展中，因此，对于某一种框架，它可能在当前有一些不支持，或者不完善的特性，但可能会在将来的版本中得到改善。

1. 框架的设计逻辑

**Caffe**：提供非常完备的基本网络模块，特别是卷积神经网络的表现非常出色。在计算机视觉领域和图像识别领域非常受欢迎。但底层的运算模块并不直接暴露给用户，因此比较适合短平快的工程性项目。

**Tensorflow**：吸收了很多开源框架的经验，既提供了底层运算接口，使得用户既可以对底层运算进行优化；同时也提供了完善的网络模块，让用户可以快速进行工程项目开发。

**Theano**：支持当前大部分的主流网络模型，它是一种符号语言，同时它也是第一种引入符号计算图来编译符号表达式的开源库。相比于其他3种框架，它更注重底层的开发，因此更适合进行性能优化和算法的研究。

**Torch**：与Caffe类似，是一个注重功能性的平台，提供了非常丰富的网络模块接口，但用户很难对底层的运算模块进行直接的操作。

2. 底层的开发语言与上层的接口语言

表 1.1 四大深度学习开源框架使用语言的比较

框 架	底层语言	提供的接口语言
Caffe	C++	C++, Python、MATLAB
Tensorflow	C++, Python	C++, Python
Theano	C、Python	Python
Torch	C、Lua	C、C++、Lua

从表1.1可以看出，4种框架都采用了高效率语言 C/C++来编写，而上层接口语言则以 C++和Python为主。这主要是由于底层的设计需要考虑运行的效率，而上层的设

计则更多会考虑到整个语言的生态,生态越丰富,覆盖人群越多,可扩展性也要更好。

### 3. 开源社区的活跃度

图1.15展示了截至2016年12月,4个深度学习框架在Github上的star、fork以及贡献者的数目,这些指标也在一定程度上反映了四者在当前开源社区的活跃度。在star和fork数目上, Tensorflow当前呈现一枝独秀的态势,而在代码贡献者数量上, Tensorflow和Theano的参与人数明显要比另外两个框架多。

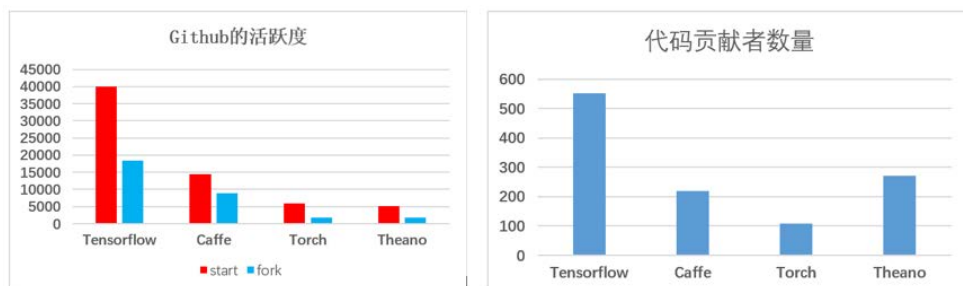


图1.15 四大深度学习开源框架在Github的活跃度与代码贡献者数量的比较

## 参考文献

- [1] R.Sutton et al. Reinforcement learning: An introduction, 1998.
- [2] Bacon, P.-L., Bengio, E., Pineau, J., and Precup, D. (2015). Conditional computation in neural networks using a decision-theoretic approach. In 2nd Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM 2015). 453.
- [3] 10 Breakthrough Technologies 2013. MIT Technology Review.
- [4] David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams. (1986a). Learning internal representations by error propagation. In parallel Distributed Processing: Explorations in the Microstructure of Cognition, volume 1, 318-362, MIT Press, Cambridge, MA.
- [5] David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams. (1986b). Learning representations by back-propagating errors. Nature, 323(9): 533-536.
- [6] Geoffrey E. Hinton, Simon Osindero, Yee-Whye The. A fast learning algorithm for deep belief nets. Neural Computation. 2006 Jul;18(7):1527-54.
- [7] Andrew Ng. Unsupervised Feature Learning and Deep Learning. 2011.
- [8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. Computer Vision and Pattern Recognition. 2015.
- [9] Bagnell, J. A. and Bradley, D. M. (2009). Differentiable sparse coding. In D. Koller, D. Schuurmans, Y.

- Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21 (NIPS'08)*, pages 113–120. 501.
- [10] Bengio, Y., A. Courville, and P. Vincent. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8): 1798-1828.
- [11] Evaluation of Deep Learning Toolkits.
- [12] Gerstner, W. and W. Kistler. (2002). *Spiking Neuron Models: Single Neurons, Populations, Plasticity*, Cambridge University Press, Cambridge, UK.
- [13] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning*, Adaptive Computation and Machine Learning series. The MIT Press. 2016. ISBN-13: 978-0262035613.
- [14] Reed R.D, R.J. Marks. (1998). *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. MIT Press, Cambridge, MA.
- [15] TensorFlow - Google's latest machine learning system, open sourced for everyone.
- [16] Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423-1447.
- [17] Tickle, A.B, R. Andrews, M. Golea, and J. Diederich. (1998). The true will come to light: Direction and challenges in extracting the knowledge embedded within trained artificial neural networks. *IEEE Transactions on Neural Networks*, 9(6): 1057-1067.

# 2

## Theano 基础

“工欲善其事，必先利其器”，人工智能领域的学习，除了要掌握必要的理论知识外，还需要参与实践，把理论知识应用到实际中，同时，在实际应用中加深对理论知识的理解。深度学习在近些年，不管是在学术界还是工业界，都取得了快速的发展，也催生了很多优秀的深度学习工具库的出现，上一章介绍了几种常用的开源工具，并简要比较了它们的特点，本章将讲解Theano的基础知识，本书的代码都是基于Theano来编写。

Theano是遵循BSD协议的一个开源项目，在2008年，由Yoshua Bengio领导的加拿大蒙特利尔理工学院LISA实验室开发。在本书编写过程中，Theano的最新版本是0.9.0，本书之所以选用Theano作为开发工具，主要基于下面两点考虑。

第一，Theano是最早的深度学习开发工具之一，也是第一个使用“符号计算图”来描述模型表达式的开源架构，当前很多的优秀开源工具库，都派生于或借鉴了Theano的底层设计，如Tensorflow、Keras等，因此，了解Theano的使用，特别是符号图的机制能帮助我们更好地学习其他的开源工具。

第二，从严格意义上来说，Theano是一个基于Python和Numpy构建的数值计算工具包，通过Theano可以定义最优化以及估值高维度的数学表达式。与一些较为成熟的商业工具库，如Tensorflow、Torch相比，它显得更为学术，此外，它并没有专门提供深度学习相关的接口，因此，用户构建模型时需要从最基本的网络层开始构建，对于深入理解模型的原理也有很大的好处。

Theano工具库包含的功能很多，涉及15个模块，如图2.1所示。

- `compile` – Transforming Expression Graphs to Functions
- `config` – Theano Configuration
- `d3viz` – d3viz: Interactive visualization of Theano compute graphs
- `gof` – Theano Internals [doc TODO]
- `gradient` – Symbolic Differentiation
- `misc.pkl_utils` – Tools for serialization.
- `printing` – Graph Printing and Symbolic Print Statement
- `sandbox` – Experimental Code
- `scalar` – Symbolic Scalar Types, Ops [doc TODO]
- `scan` – Looping in Theano
- `sparse` – Symbolic Sparse Matrices
- `sparse` – Sparse Op
- `sparse.sandbox` – Sparse Op Sandbox
- `tensor` – Types and Ops for Symbolic numpy
- `typed_list` – Typed List

图2.1 Theano工具库包含的模块（摘自Theano官方文档）

限于本书的篇幅和写作目的，我并没有打算将本章作为API文档讲解，也不会深入分析Theano的底层架构原理，本章的重点是对Theano的基础知识和编程范式进行讲解，读者在完成本章的学习后，应该能够使用Theano进行独立简单的深度学习开发。读者如果对本章的内容比较熟悉，可以跳过本章，如果想更全面地理解Theano的架构，或某一个模块和接口的使用，也可以查阅Theano的官方文档。

## 2.1 符号变量

使用编程语言进行编程时，需要用到各种变量来存储各种数据信息，Theano 是基于 Python 和 Numpy 实现的数值计算工具库，严格来说并不是一门独立的编程语言，但 Theano 有其独立的变量体系，Theano 的变量类型称为符号变量，记为 `TensorVariable`，它是Theano表达式和运算操作的基本单元。Theano创建符号变量的方式主要有下面几种。

### 1. 使用内置的变量类型创建

Theano 当前支持 7 种内置的变量类型，分别是 `scalar`、`vector`、`row`、`col`、`matrix`、`tensor3`、`tensor4`。以创建向量类型为例，看看如何使用内置方法定义向量类型变量。



```
>>> import theano
>>> import theano.tensor as T
>>> data = T.vector(name='data', dtype='float32')
```

`vector`函数需要指定下面两个参数。

- `name`: 指定变量的名字。
- `dtype`: 指定变量的数据类型。Theano变量支持的数据类型包括8种, `int8`、`int16`、`int32`、`int64`、`float32`、`float64`、`complex64`、`complex128`。

同理, 要创建其他类型的变量, 只需要将`vector`替换为其他类型既可, 比如要创建矩阵类型, 可以使用`T.matrix`。

## 2. 自定义变量类型

内置的变量类型只能处理四维及以下的变量。当需要处理更高维的数据时, 我们就需要自定义变量类型, Theano 提供了 `TensorType` 方法来自定义数据类型。

```
>>> import theano
>>> import theano.tensor as T
>>> mytype = T.TensorType(dtype, broadcastable, name=None, sparse_grad=False)
```

`TensorType`函数需要指定4个参数, 其中`dtype`和`broadcastable`是必须指定的, 也是最常用的参数。

- `name`: 指定变量的名字。
- `dtype`: 指定变量的类型, 参考前面内置变量的定义。
- `broadcastable`: 是一个由`true`或`false`值构成的布尔类型元组, 元组的大小等于变量的维度大小, 如果元组中的某一元素值为`true`, 则表示变量在对应维度上的数据可以进行广播 (`broadcast`), 否则数据不能广播。

`broadcast`机制是很多数值计算库都会实现的一种机制, 因为在做线性代数运算时, 经常会出现不同维度大小的数据类型运算, 比如一个标量数据与一个矩阵相加, 一个向量与一个矩阵相加等。如果没有广播的机制, 需要先把低维的数据变换到高维, 当两者的维度大小一致的时候, 才能进行相应的操作符运算, 而广播机制则可以直接执行这种异构数据类型的运算操作, 不需要经过烦琐的维度转换。图2.2展示了一个向量与矩阵相加的时候, 广播机制的内部转化过程。

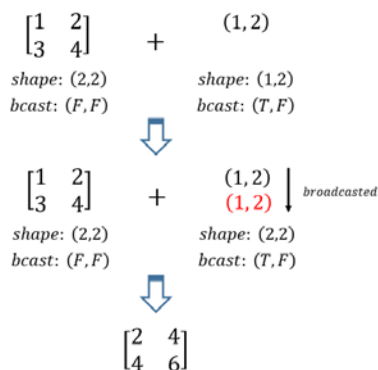


图2.2 broadcast机制的运算

下面是用TensorType方法创建的一个五维张量类型，我们将其broadcastable设置为(False,)\*5，表示新创建的类型mytype在5个维度上都不支持广播机制。

```
>>> import theano
>>> import theano.tensor as T
>>> mytype = T.TensorType('float32', (False,)*5)
>>> data = mytype('x')
>>> data.type()
<TensorType(float32, 5D)>
```

### 3. 将Python类型变量或者Numpy类型变量转化为Theano共享变量

共享变量是Theano实现变量更新的重要机制，我们将在后面详细讲解Theano的共享变量机制。要创建一个共享变量，只需要把一个Python对象或Numpy对象传递给shared函数。

```
>>> import theano
>>> import theano.tensor as T
>>> import numpy
>>> data = numpy.array([[1, 2, 3], [4, 5, 6]])
>>> shared_data = theano.shared(data)
>>> type(shared_data)
<class 'theano.tensor.sharedvar.TensorSharedVariable'>
```

### 4. 将Python类型变量或者Numpy类型变量转化为Theano变量

如果变量不需要更新，那么Python或Numpy类型将转化为普通的Theano变量。一般来说，当表达式既含有Theano变量，又含有Python或Numpy类型变量时，编译器会自动将Python或Numpy类型变量转换为Theano变量。如果想要手动变换，Theano也提供了相应的方法。

- *theano.tensor.constant*：将参数类型转化为Theano常量。

- `theano.tensor.as_tensor_variable`: 将参数类型转化为Theano变量。

## 2.2 符号计算的抽象——符号计算图模型

2.1节详细讲解了符号变量（Tensorvariable）的定义方法，本节将讲解符号计算图的概念。要定义一条符号表达式，首先按照2.1节介绍的方式创建表达式所需要用到的变量，然后把这些符号变量通过操作符（op）结合在一起。

Theano处理符号表达式时是通过把符号表达式转换为一个计算图（graph）来处理，因此，理解计算图的基本原理和底层的工作机制，对于编写和调试Theano代码有很大的帮助。本节来探讨Theano的符号计算图的知识。

符号计算图的节点由下面4种类型节点构成：variable、type、apply和op。

（1）**variable节点**：即符号变量节点，符号变量是符号表达式中存放信息的数据结构，可以分为输入符号变量和输出符号变量。

```
>>> import theano
>>> x = theano.tensor.ivector('x')
>>> y = x ** 2
```

定义一个符号表达式 $y = x ** 2$ ，其中 $x$ 和2均为输入符号变量，而 $y$ 则是输出符号变量，一个符号变量具有下面4个重要的字段。

1) **type**: 指向type节点，我们将在后面讲解type节点的相关知识。

2) **owner**: 可以是None或者指向一个apply节点，有关apply节点的知识将在后面讲解。

3) **index**: 一个索引值，当owner不为None时，如果变量是输入符号变量，则表示该变量在owner所指向的符号表达式中是第index个输入变量；相反，如果变量是输出符号变量，则表示该变量在owner所指向的符号表达式中是第index个输出。

4) **name**: 定义了变量的名字。

（2）**type节点**: 当定义了一种具体的变量类型以及变量的数据类型时，Theano为其指定了数据存储的限制条件。

```
>>> x = theano.tensor.irow('x')
```

定义了一个变量 $x$ ，其变量类型是row，变量的数据类型是int32（i是int32的缩写），

这样Theano为变量 $x$ 指定了下面的限制条件，后面为 $x$ 指定具体的数值时，如果不满足这些约束，将会产生TypeError错误。

- 1) 底层必须以numpy.ndarray作为数据结构。
- 2) 数据类型必须是int32，则 $x$ 是一个int32的整数数组。
- 3) 变量的形态大小必须为 $(1, n)$ ，则第一维的大小必须为1。

(3) **apply节点**：表示把某一种类型的符号操作符应用到具体的符号变量(variable)中。与variable节点不同，apply节点无须由用户指定，前面我们已经指出，apply节点是通过输出符号变量的owner字段来获取，一个apply节点包括3个重要的字段。

- 1) op：指向符号表达式的操作符节点。
- 2) inputs：符号表达式的输入参数变量列表。
- 3) outputs：符号表达式的输出结果变量列表。

(4) **op节点**：操作符节点，等价于编程语言中的函数定义，op定义了一种符号变量间的运算，以某种类型的符号变量作为输入，输出另一种符号变量，如+、-、sum()、tanh()等。

这里要把op的定义和apply的定义区分开来，从编程语言的角度来说，op等价于函数定义，而apply等价于函数的应用。通过下面的例子来考察两者之间的区别。

如果用Python的语法来理解Theano的计算图结构，假设定义了一个函数fun，那么将产生一个op节点fun。

```
>>> def fun(x): ...
```

如果在代码中调用了该函数，那么将产生一个apply节点，并且该节点的op字段将指向fun节点。

```
>>> a = fun(5)
```

下面通过一个具体的例子来理解符号计算图的结构。

```
>>> import theano.tensor as T
>>> x = T.dmatrix('x')
>>> y = T.dmatrix('y')
>>> z = x + y
```

上面的代码逻辑非常简单，定义了两个矩阵变量 $x$ 和 $y$ ，定义符号表达式 $z = x + y$ ，

该符号表达式转化为对应的符号计算图结果如图2.3所示。

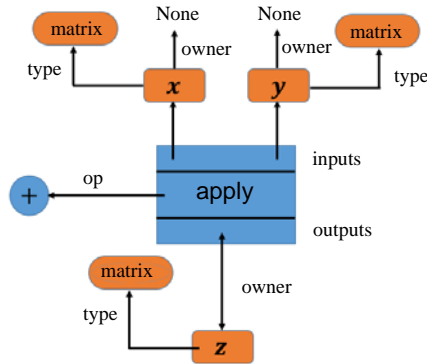


图2.3 符号计算图

我们可以自行验证每一个节点对应的字段结果输出。

如想要获取`apply`节点，通过输出符号变量`z`的`owner`字段。

```
>>> z.owner
Elemwise{add,no_inplace}(x, y)
```

如果想要获取表达式的所有输入符号变量和所有输出符号变量，可以通过`apply`节点的`inputs`和`outputs`字段。

```
>>> z.owner.inputs
[x, y]
>>> z.owner.outputs
[Elemwise{add,no_inplace}.0]
```

在上面的例子中，表达式是两个矩阵相加，形式比较简单，但对于复杂的表达式或函数，要画出完整的符号计算图是一件很麻烦的事情，为此Theano支持把计算图打印到终端或输出到外部文件，要打印符号计算图，首先需要定义`printing`模块。

输出到终端支持两种模式：`pp`模式和`debugprint`模式。`pp`模式的输出结果简洁紧凑，就像数学表达式一样。而`debugprint`模式，顾名思义，是`debug`常用的模式，因此它的输出更加详细，下面分别使用`pp`模式和`debugprint`模式来查看`z`的结果。

```
>>> theano.printing.pprint(z)
'(x + y)'
>>> theano.printing.debugprint(z)
Elemwise{add,no_inplace} [id A] ''
|x [id B]
|y [id C]
>>>
```

通常采用 Theano 自带的 `pydotprint` 接口来输出到外部文本。`pydotprint` 接口有两

个重要的参数：`fct` 和 `outfile`，其中 `fct` 表示待打印的函数或变量列表，`outfile` 表示输出的文件名。考查符号表达式  $z=x+y$ ，使用 `pydotprint` 打印的符号计算图结果如图2.4所示，读者也可以对比图2.4与图2.3，观察它们之间的异同。

```
>>> theano.printing.pydotprint(z, outfile="out.png")
```

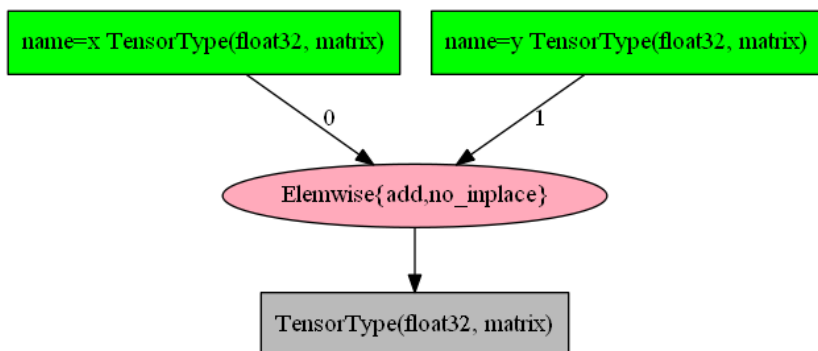


图2.4 pydotprint打印结果

## 2.3 函数

函数是Theano的一个核心设计模块，前面已经讲解了Theano如何把一个符号表达式转化为符号计算图，函数的作用则是为我们提供了一个接口，把符号计算图编译为可调用的函数对象。

### 2.3.1 函数的定义

首先来考察Theano函数的定义语法，函数定义最为重要的4个参数分别是：`inputs`、`outputs`、`updates`、`givens`。

```
>>> theano.function(inputs, outputs, updates, givens)
```

(1) **inputs**：用于指定函数的输入变量列表。python 以列表的形式来表示，列表的每一个元素是一个In类型，In类型的构造函数有很多参数设置，而最常用的是下面两个参数。

- 1) **variable**：指定符号变量。
- 2) **value**：指定变量的默认值。

更详细的参数定义和设置可以参考Theano的官方文档。

(2) **outputs**: 指定函数的输出变量列表。如果为空, 则没有输出; 也可以只有一个输出变量; 或者以列表的形式来表示多个输出变量。如果不为空, 则每一个输出元素是一个Out类, Out类的构造函数相对简单, 其中我们一般只需要指定输出的符号标量即可。

(3) **updates**: 指定共享变量的更新策略, 通常以字典或元组列表的形式来指定。updates应用最广泛的就是在最优化计算过程中, 指定每一次迭代时参数的更新策略, 下面的代码展示了在梯度下降算法中使用updates来对权重参数进行迭代更新的过程。

```
>>> import theano
>>> params = [...] # 共享变量参数列表
>>> cost = f(params) # 损失函数
>>> gparams = theano.tensor.grad(cost, params)
>>> updates = [(p, p - lr*gp) for p, gp in zip(params, gparams)]
```

(4) **givens**: 是一个字典或者元组列表, 记为[(var1, var2)], 表示在每一次函数调用时, 在符号计算图中, 把符号变量var1节点替换为var2节点, 该参数特别适用来指定训练数据集的 batch 大小。

## 2.3.2 Logistic回归

以Logistic回归为例子考察函数的定义和使用, 本节只讲解如何使用Theano来实现Logistic回归, 目的是对前面的内容做一个总结。本节并不会深入解析Logistic回归的原理, 有关Logistic回归的相关知识将在第6章讲解。

Logistic回归是机器学习中常用的二元分类算法, 设现有训练数据集 $D = (X, Y)$ , 其中 $Y$ 是训练数据对应的label,  $X$ 的大小为 $N \times M$ ,  $Y$ 的大小为 $N \times 1$ , 其中 $N$ 为样本总数,  $M$ 为特征数。

```
>>> N = 10000
>>> M = 784
>>> X = numpy.random.randn(N, M)
>>> Y = numpy.random.randint(low=0, high=2, size=N)
```

Logistic回归通过使用Logistic函数来作为预测模型。

$$f(z) = \frac{1}{1 + e^{-z}} \quad (2.1)$$

其中 $z = w_1x_1 + w_2x_2 + \dots + w_Mx_M + b$ 是一个线性模型,  $w = (w_1, w_2, \dots, w_M)$ 和 $b$

是待确定的参数。

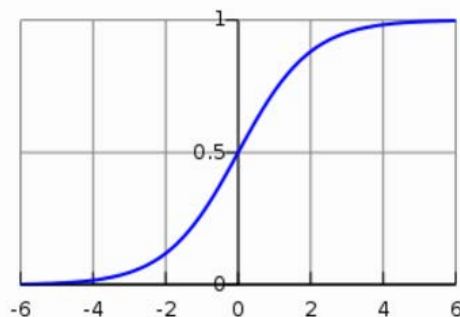


图2.5 Logistic函数

Theano把参数初始化为共享变量，以便迭代时能进行更新。

```
>>> w_value = numpy.random.randn(M)
>>> w = theano.shared(value=w_value, name='w')
>>> b = theano.shared(value=0.0, name='b')
```

当参数 $w$ 和 $b$ 确定后，对于输入数据 $(x_1, x_2, \dots, x_M)$ ，将其代入Logistic函数，Logistic函数是一个值为  $0 \leq y \leq 1$  的函数，当 $y \leq 0.5$ 时，预测值为0；反之，若 $y > 0.5$ 时，预测值为1。

```
>>> x = T.dmatrix('x')
>>> y = T.ivector('y')
>>> output = 1.0 / (1.0 + T.exp(-T.dot(x, w) - b))
>>> predict = output > 0.5
...
```

下一步要指定损失函数，Logistic回归采用对数似然损失函数（交叉熵）作为最优化的目标函数：

$$\text{cost} = -\frac{1}{k} \sum_{i=1}^k (y_k \times \ln p(y_k) + (1 - y_k) \times \ln(1 - p(y_k))) \quad (2.2)$$

其中 $k$ 是训练数据集的数量， $p(y_k)$ 是第 $k$ 个训练数据的label值取 $y_k$ 的概率， $y_k$ 的取值只能取0或1，可以加上正则化项来防止过拟合，最后的损失函数公式为：

$$\text{cost} = -\frac{1}{k} \sum_{i=1}^k (y_k \times \ln p(y_k) + (1 - y_k) \times \ln(1 - p(y_k))) + \gamma \times \sum_{i=1}^M w_i^2 \quad (2.3)$$

```
>>> loss = - y*T.log(output) - (1 - y)*T.log(1-output)
>>> cost = T.mean(loss) + 0.01 * (w**2).sum()
>>> gw, gb = T.grad(cost, [w, b])
```



最后定义训练函数，mini-batch 梯度下降法来训练模型，通过givens参数来指定每一次迭代的训练样本数据。

```
>>> train = theano.function(
    inputs = [index],
    outputs = cost,
    updates = [(w, w-0.1*gw), (b, b-0.1*gb)],
    givens = {x : X[index*batch_size : (index+1)*batch_size],
              y : Y[index*batch_size : (index+1)*batch_size]})
```

将上面的步骤进行归纳总结，可以得到完整代码，如下所示。

```
>>> import numpy
>>> import theano
>>> import theano.tensor as T
>>> N, M = 1000, 78
>>> X = numpy.random.randn(N, M)
>>> Y = numpy.random.randint(low=0, high=2, size=N)
>>> train_X = theano.shared(value = X)
>>> train_Y = theano.shared(value = Y)
>>> batch_size = 100
>>> w_value = numpy.random.randn(M)
>>> w = theano.shared(value=w_value, name='w')
>>> b = theano.shared(value=0.0, name='b')
>>> x = T.dmatrix('x')
>>> y = T.ivector('y')
>>> index = T.iscalar('index')
>>> output = 1.0/(1.0 + T.exp(-T.dot(x, w)-b))
>>> predict = output > 0.5
>>> loss = - y*T.log(output) - (1-y)*T.log(1-output)
>>> cost = T.mean(loss) + 0.001 * (w**2).sum()

>>> gw, gb = T.grad(cost, [w, b])
>>> train = theano.function(
    inputs = [index],
    outputs = cost,
    updates = [(w, w-0.1*gw), (b, b-0.1*gb)],
    givens = {x : train_X[index*batch_size : (index+1)*batch_size],
              y : train_Y[index*batch_size : (index+1)*batch_size]}
)

>>> batch = N/batch_size
>>> for n_epoch in range(100):
    ret = []
    for batch_index in range(batch):
        ret.append(train(batch_index))
    print("epoch: %d, average cost is %lf" % (n_epoch, numpy.mean(ret)))
```

### 2.3.3 函数的复制

在实际应用中会遇到一种情况，多个函数之间的功能相似，只是参数不一样，比如我们用同一个算法对不同的模型进行训练，不同的模型训练的参数不一样，但是算法是相同的，这个时候可以使用函数的复制功能，将一个函数复制给另一个函数后，这两个函数之间具有独立的计算图结构，相互之间并不会产生影响。Theano函数的复制

通过copy函数来实现。

以累加器为例，下面的函数通过定义一个共享变量state来作为累加变量，每一次调用函数accumulator时，state的值都会发生变化。

```
>>> import theano
>>> import theano.tensor as T
>>> state = theano.shared(0)
>>> inc = T.iscalar('inc')
>>> accumulator = theano.function([inc], state, updates=[(state, state+inc)])
```

调用accumulator函数，查看输出结果。

```
>>> accumulator(10)
array(0)
>>> state.get_value()
array(10)
```

新建另一个函数new\_accumulator，它实现的功能与accumulator函数完全相同，但累加的变量不一样。new\_accumulator是定义在new\_state上的累加函数，通过copy函数来实现这个功能，通过swap参数来交换两个共享变量。

```
>>> new_state = theano.shared(0)
>>> new_accumulator = accumulator.copy(swap={state:new_state})
```

来验证一下结果。

```
>>> new_accumulator(100)
[array(0)]
>>> new_state.get_value()
array(100)
>>> state.get_value()
array(10)
```

正如我们所预料的，调用new\_accumulator函数并没有影响原来的state变量。

如果只想在原来函数的基础上去除共享变量的更新，同样可以通过copy函数来实现这个功能，通过delete\_updates参数来实现这个功能。

```
>>> null_accumulator = accumulator.copy(delete_updates=True)
```

检验输出结果。

```
>>> null_accumulator(9000)
[array(10)]
>>> state.get_value()
array(10)
```

可以看到调用null\_accumulator函数并没有影响state变量，事实上，这个函数每次调用都会输出相同的结果。

## 2.4 条件表达式

Theano是一种符号语言，因此不能直接使用Python的if语句，可以考察下面的语句。

```
>>> import theano
>>> import theano.tensor as T
>>> a = T.scalar('a')
>>> b = 0.1 if a==0 else a
>>> b.eval({a:0})
array(0.0, dtype=float32)
>>> b.eval({a:1})
array(1.0, dtype=float32)
```

可以看到，不管 $a$ 的值取多少， $b$ 的值均为 $a$ ，这是由于 $a$ 是一个符号变量， $a == 0$ 的返回结果永远是False。要对符号变量进行比较，需要使用符号变量的条件表达式，当前Theano支持ifelse和switch两种表达式来进行符号变量之间的比较。

(1) ifelse函数接收3个参数，分别是布尔型的条件值，以及两个分支的输出结果。

```
>>> import theano
>>> import theano.tensor as T
>>> from theano.ifelse import ifelse
>>> ifelse(condition, express1, express2)
```

(2) switch函数接收3个参数，分别是tensor类型变量，以及两个分支的输出结果。

```
>>> import theano
>>> import theano.tensor as T
>>> T.switch(condition, express1, express2)
```

下面考察ifelse与switch的区别，从上面可以看出，两者的使用方法非常接近，当条件表达式condition的值为1时，表示真值，将执行第二个参数，即表达式express1，否则执行第三个参数表达式express2。两者之间主要存在下面两个差异。

- ifelse的条件表达式condition只支持标量值，而switch的条件表达式可以是任意的Theano符号变量，而不只是标量值。因此，switch的使用范围要比ifelse更广。
- ifelse的运算是惰性的，当条件表达式condition的值为1时，就执行express1，不会执行express2。但switch则会把两个表达式都执行，当express1和express2都执行完毕后，才根据条件表达式condition的值返回express1或express2。通过下面的例子来考察两者的区别。

```
>>> import theano
>>> import theano.tensor as T
>>> r, s, t = T.iscalars('r', 's', 't')
>>> r_ = theano.printing.Print('r')(r)
>>> s_ = theano.printing.Print('s')(s)
>>> u = T.switch(T.eq(t, 0), s_, r_)
```

这里定义了一个switch条件表达式，并利用print语句来跟踪语句的执行次序，结果如下所示。

```
>>> u.eval({t: 0, r: 1, s: 2})
s __str__ = 2
r __str__ = 1
array(2)
>>> u.eval({t: -1, r: 1, s: 2})
s __str__ = 2
r __str__ = 1
array(1)
```

switch条件表达式会执行r\_和s\_，最后根据condition返回结果数据。ifelse表达式如下所示。

```
>>> import theano
>>> import theano.tensor as T
>>> from theano.ifelse import ifelse
>>> r, s, t = T.iscalars('r', 's', 't')
>>> r_ = theano.printing.Print('r')(r)
>>> s_ = theano.printing.Print('s')(s)
>>> u = ifelse(T.eq(t, 0), s_, r_)
```

这里定义的ifelse条件表达式与前面switch条件表达式完全一样，考察它的结果输出。

```
>>> u.eval({t: 0, r: 1, s: 2})
s __str__ = 2
array(2)
>>> u.eval({t: -1, r: 1, s: 2})
r __str__ = 1
array(1)
```

可以看出，ifelse的执行过程是一种“短路”策略，只有其中一个分支会被执行。

## 2.5 循环

循环是程序设计语言中很重要的一个模块，Theano的循环操作使用scan模块来实现，类似于Python的for语句，首先来考察 scan 的函数声明。

```
>>> theano.scan(fn, sequences=None, outputs_info=None, non_sequences=None, n_steps=None, truncate_gradient=-1, go_backwards=False, mode=None, name=None, profile=False, allow_gc=None, strict=False)
```

scan的参数很多，读者如果想要了解每一个参数的详细信息，可以查阅官方的文档，这里考察常用的几个参数。

(1) **sequences**: 是一个由Theano变量或字典构成的列表，它们的值将作为参数传递给函数fn。列表中的每一个元素都是一个序列，每一次迭代可以传递序列的一个元素或者多个元素。下面通过一个例子来理解sequences的使用。

```
>>> theano.scan( ..... ,
                  sequences=[dict(input=sequence1, taps=[-3, -1]),
                              sequence2,
                              dict(input=sequence3)],
                  .....)
```

sequences参数包括了sequence1、sequence2、sequence3三个输入序列。

1) sequence1: 以字典的形式来表示，当用字典来表示时，字典中可以包括input和taps两个key，其中input是输入序列，taps是索引，上面的代码表示在第t步迭代时，sequence1传递给fn的参数有sequence1[t - 3]和sequence1[t - 1]。

2) sequence2: 以普通的Theano变量形式传递，事实上，它等价于下面的代码。

```
>>> dict(input=sequence2, taps=[0])
```

当忽略taps参数时，Theano默认会自动添加taps = [0]，因此，在第t次迭代时，sequence2传递给fn的参数为sequence2[t]。

3) sequence3: 可以参考sequence1和sequence2来理解。

(2) **outputs\_info**: 与sequences的表达相似，outputs\_info同样是一个由Theano变量或字典构成的列表，列表中的每一个元素是函数fn的输出结果的初始值。用一个同样的例子来说明。

```
>>> theano.scan(
    .....,
    outputs_info = [dict(initial=sequence1, taps=[-1, -2]),
                    sequence2,
                    dict(initial=sequence3)],
    .....)
```

为了方便与sequences比较，我们在outputs\_info中放置了3个相同名字的变量元素sequence1、sequence2和sequence3，读者可以比较两者之间定义的差异。

1) sequence1: 以字典的形式来表示，outputs\_info用字典来表示的时候，可以包括initial和taps两个key，initial用来定义初始值，taps是索引。上式表示在第t步迭代时，sequence1传递给fn的参数有sequence1[t - 1]和sequence1[t - 2]。

2) **sequence2**: 以普通的Theano变量形式传递，它等价于下面的代码。

```
>>> dict(initial=sequence2, taps=[-1])
```

当忽略taps时，默认会自动添加taps = [-1]，因此，在第t次迭代，sequence2传递给fn的参数为sequence2[t - 1]。

3) **sequence3**: 可以参考sequence1和sequence2来理解。

(3) **non\_sequences**: 是一个不变量或常数值列表，与outputs\_info和sequences不同的是，non\_sequences参数在每一次迭代时都是全部传送给迭代函数，在实际应用中，一般是把non\_sequences设置为模型的权重参数列表。

(4) **fn**: 是scan最核心的部分，它是一个函数，定义了每一次循环的处理逻辑，fn既可以用lambda匿名函数来表示，也可以是自定义函数来实现。需要特别注意的是，它对函数参数的定义顺序，以及函数的输出有严格的要求。

首先，fn的输入参数是由前面提到的3个参数来提供，不接受额外的输入，参数定义顺序为：sequences、outputs\_info、non\_sequences，看看下面的代码块。

```
>>> theano.scan(
    fn = lambda a1, a2, a3, a4, a5, a6, a7, a8, a9: ... ,
    sequences = [dict(input=sequence1, taps=[-3, -2, -1]),
                  sequence2],
    outputs_info = [dict(initial=sequence3, taps=[-1, -2]),
                    sequence4],
    non_sequences = [arg1, arg2],
    .....
)
```

lambda函数一共有9个参数，这些参数对应的值如表2.1所示。

表 2.1 lambda 函数的参数及其对应值

参数	值
a1	sequence1[t-3]
a2	sequence1[t-2]
a3	sequence1[t-1]
a4	sequence2[t]
a5	sequence3[t-1]
a6	sequence3[t-2]
a7	sequence4[t-1]
a8	arg1
a9	arg2

(5) **n\_steps**: 用来指定迭代的次数，sequences与n\_steps两者至少要存在一个，

否则scan无法知道迭代的步数。

(6) **truncate\_gradient**: 这是一个专为循环神经网络训练设计的参数。利用scan来实现BPTT时, truncate\_gradient设置了向前传播的长度, 当值为-1时, 表示采用的是传统的BPTT算法, 但如果truncate\_gradient的值大于0, 表示向前执行完truncate\_gradient步时, 会提前结束返回, 这是一种截断策略。在第12章中会看到, 传统的BPTT会导致梯度消失问题, 截断是防止梯度消失的一种有效策略。

(7) **strict**: 当设置为true, 必须保证把所有用到的Theano共享变量都放置在non\_sequences参数中。

下面通过几个具体的例子来介绍scan的使用方法。

- 示例一: 计算 $A^k$ 。

任务描述:

计算 $A$ 的 $k$ 次方, 其中 $A$ 可以是实数、向量或者矩阵。

函数设计:

如果使用Python来完成上述的功能, 可以写出下面的伪代码。

```
>>> result = 1
>>> for i in range(k):
    result = result * A
```

分析上面的伪代码, 要完成 $A^k$ 这个功能, 我们需要做到以下几个方面: 第一是初始值, 在最开始把初始值result设为1。第二是常量数据A。第三是迭代次数k, 告知循环迭代的次数。

与上面的思想类似, Theano采用scan来实现循环的操作, 首先把结果初始值放在outputs\_info中, 常量值则对应于non\_sequences, 迭代的步数 $k$ 对应于n\_steps。最后是迭代函数fn的设计, 在本例中, fn的实现较为简单, 直接使用匿名函数来实现, 函数fn的参数分别对应outputs\_info和non\_sequences, 代码如下所示。

```
>>> import theano
>>> import theano.tensor as T
>>> k = T.iscalar('k')
>>> A = T.iscalar('A')
>>> outputs, updates = theano.scan(
    fn = lambda x, A : x*A,
    outputs_info = T.ones_like(A),
    non_sequences = [A],
    n_steps = k
)
>>> fun = theano.function(inputs = [k, A], outputs = outputs[-1], updates = updates)
```

验证结果：

计算 $5^3$ ，其中 $A = 5$ ， $k = 3$ 。

```
>>> fun(3, 5)
array(125)
```

计算 $8^2$ ，其中 $A = 8$ ， $k = 2$ 。

```
>>> fun(2, 8)
array(64)
```

- 示例二：多项式计算。

任务描述：

给定多项式的系数列表 $(a_0, a_1, a_2, \dots, a_m)$ 和未知项 $x$ ，计算多项式 $f = a_0 + a_1 \times x + a_2 \times x^2 + \dots + a_m \times x^m$ 的值。

函数设计：

Sequences：包括多项式系数列表 $(a_0, a_1, a_2, \dots, a_m)$ 和对应的指数项索引列表 $(0, 1, 2, \dots, m)$ 。

outputs\_info：输出的初始值，我们设置为0.0。

non\_sequences：在每一次迭代时，未知项参数 $x$ 是不变的，因此把 $x$ 放置在non\_sequences中。

代码示例：

```
>>> import theano
>>> import theano.tensor as T
>>> a = T.vector('a', dtype=theano.config.floatX)
>>> x = T.scalar('x', dtype=theano.config.floatX)
>>> outputs, updates = theano.scan(
    fn = lambda a, p, o, x : o + a*(x ** p),
    sequences = [a, T.arange(a.shape[0]).astype(theano.config.floatX)],
    outputs_info = T.as_tensor_variable(0.0).astype(theano.config.floatX),
    non_sequences = [x]
)
>>> fun = theano.function(inputs = [x, a], outputs = outputs[-1], updates = updates)
```

验证结果：

计算 $f = 1 + 3 \times x^2 - 2 \times x^3 + 2 \times x^4$ ，其中 $x$ 的值为2.0。

```
>>> fun(2.0, [1, 0, 3, -2, 2])
array(29.0, dtype=float32)
```

计算 $f = 2 \times x + 3 \times x^3 + 5 \times x^4$ ，其中 $x$ 的值为2.5。



```
>>> fun(2.5, [0, 2, 0, 3, 5])
array(247.1875, dtype=float32)
```

- 示例三：数据替换。

### 任务描述：

给定一个索引矩阵 $[(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)]$ 和待替换的值向量 $[v_1, v_2, \dots, v_m]$ ，把一个矩阵 $\text{matrix}$ 对应的位置元素 $\text{matrix}[x_i, y_i]$ 替换为 $v_i$ 。

### 函数设计：

**Sequences**：包括索引列表和替换数据列表。索引列表用矩阵来表示，每一行是一个位置元组 $(x_i, y_i)$ 。

**outputs\_info**：不需要输出信息，无须依赖上一步的结果输出。

### 代码示例：

```
>>> import theano
>>> import theano.tensor as T
>>> idx = T.imatrix('idx')
>>> v = T.ivector('v')
>>> model = T.imatrix('model')
>>> def _step(idx, v, model):
>>>     return T.set_subtensor(model[idx[0], idx[1]], v)

>>> outputs, updates = theano.scan(
>>>     fn = _step,
>>>     sequences = [idx, v],
>>>     outputs_info = model
>>> )
>>> fun = theano.function(inputs = [idx, v, model], outputs = outputs, updates = updates)
```

### 验证结果：

设原始数据为：

$$\text{model} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

现在将 $\text{model}[1,1]$ 、 $\text{model}[0,2]$ 、 $\text{model}[1,0]$ 分别替换为 $-1$ 、 $-2$ 、 $-3$ 。

```
>>> fun([[1, 1], [0, 2], [1, 0]], [-1, -2, -3], [[1,2,3], [4,5,6], [7,8,9]])
array([[ 1,  2,  3],
       [ 4, -1,  6],
       [ 7,  8,  9]],

      [[ 1,  2, -2],
       [ 4, -1,  6],
       [ 7,  8,  9]],

      [[ 1,  2, -2],
       [-3, -1,  6],
       [ 7,  8,  9]])
```

- 示例四：条件终止。

### 任务描述：

实现break的功能，提前终止循环。

### 函数设计：

条件终止的重点在于在循环函数中，除了返回正常的输出结果之外，还需要把循环终止的条件放置在theano.scan\_module.until模块中。当输出结果大于 max 值时，就提前终止循环。

### 代码示例：

```
>>> import numpy
>>> import theano
>>> import theano.tensor as T
>>> max_value = T.iscalar()
>>> def _step(e, max_value):
>>>     return e**2, theano.scan_module.until(e**2 > max_value)

>>> outputs, updates = theano.scan(
>>>     fn = _step,
>>>     sequences = [T.arange(10)],
>>>     outputs_info = None,
>>>     non_sequences = [max_value]
>>> )
>>> fun = theano.function(inputs=[max_value], outputs=outputs, updates=updates)
```

### 验证结果：

对于序列[0,1,2,...,9]，计算每个元素的平方值，当值> 45时，循环终止退出。

```
>>> fun(45)
array([ 0,  1,  4,  9, 16, 25, 36, 49], dtype=int64)
```

正如我们预料的，循环在计算完7<sup>2</sup>后将自动退出。

## 2.6 共享变量

共享（shared）变量是实现机器学习算法参数更新的重要机制。对于一般的符号变量，它没有赋予任何的初始值。在编写深度学习程序时，需要对权重参数进行初始化，这时候需要一种带有初始值的Tensorvariable，这种带有初始值的符号变量称为共享变量，共享变量的初始值由Numpy数组指定。

创建共享变量有深拷贝模式和浅拷贝模式。深拷贝是指把Numpy数组进行深度复制，因此，后续对Numpy数组的改变不会影响到这个共享变量；浅拷贝并没有拷贝Numpy数组到共享变量中，而是为共享变量设置一个指向Numpy数组的指针，因此，后续对Numpy数组的改变会影响到这个共享变量的值。深拷贝模式和浅拷贝模式是通过borrow参数来设置，默认情况下borrow = False表示深拷贝。

```
>>> import numpy, theano
>>> np_array = numpy.ones(2, dtype='float32')
>>> s_default = theano.shared(np_array)
>>> s_false = theano.shared(np_array, borrow=False)
>>> s_true = theano.shared(np_array, borrow=True)
```

上面的代码块中，设置了三个共享变量s\_default、s\_false、s\_true，其中s\_default与s\_false是深拷贝，而s\_true使用的是浅拷贝，初始值为array([1,1])。改变np\_array的值，考察3个共享变量的值的变化情况。

```
>>> np_array += 1
>>> s_default.get_value()
array([ 1.,  1.], dtype=float32)
>>> s_false.get_value()
array([ 1.,  1.], dtype=float32)
>>> s_true.get_value()
array([ 2.,  2.], dtype=float32)
...
```

对于深拷贝的s\_default和s\_false变量来说，改变np\_array的值并没有对共享变量产生影响，但对于浅拷贝的s\_true变量来说，它的值与np\_array的值保持同步变化。

## 2.7 配置

当使用import theano导入Theano模块时，系统会自动加载Theano运行代码所需要的环境和配置，例如，使用GPU还是CPU来运行代码，使用何种编译模式等。在不同的工程项目中，可能需要不同的环境设置，因此，本节简要介绍如何手动配置Theano的属性。

Theano的所有属性通过config模块来管理，它们控制着Theano的执行行为，在命令行窗口中输入`theano.config`可以查看Theano当前的配置信息。

```
>>> print(theano.config)
floatX (('float64', 'float32', 'float16'))
    Doc: Default floating-point precision for python casts.

Note: float16 support is experimental, use at your own risk.
    Value: float32

warn_float64 (('ignore', 'warn', 'raise', 'pdb'))
    Doc: Do an action when a tensor variable with float64 dtype is created. They can't be run
on the GPU with the current(old) gpu back-end and are slow with gamer GPUs.
    Value: ignore

cast_policy (('custom', 'numpy+floatX'))
    Doc: Rules for implicit type casting
    Value: custom

int_division (('int', 'raise', 'floatX'))
    Doc: What to do when one computes x / y, where both x and y are of integer types
    Value: int

device (cpu, gpu*, opencl*, cuda*)
    Doc: Default device for computations. If gpu*, change the default to try to move computati
on to it and to put shared variable of float32 on it. Do not use upper case letters, only lower
case even if NVIDIA use capital letters.
```

设置Theano的配置属性有3种方法，按照优先级从高到低可以分为：

- (1) 通过`theano.config.property`来设置。
- (2) 通过设置`THEANO_FLAGS`环境变量。
- (3) 通过`.theanorc`文件来设置。

对于第一种方式，通常来说，`config`模块下的属性大部分都是只读的，不应该在用户的逻辑代码中改变，即使属性值是可写的，也不建议直接修改`property`，因为这样做会使代码变得难以维护，因此，本章将重点讲解后面的两种配置方式。

### 2.7.1 通过THEANO\_FLAGS配置

通过设置`THEANO_FLAGS`环境变量来改变Theano的配置，这种方式既可以是全局的，也可以仅针对某一个脚本文件。`THEANO_FLAGS`以字符串的形式显示，字符串是以逗号为分隔符的(key,value)对，如下所示。

```
>>> THEANO_FLAGS='floatX=float32,device=gpu0'
```

上面设置了`floatX`的类型是`float32`，运行的设备是`gpu0`(后面的数字是使用的GPU芯片，在多GPU的环境下，GPU设备数可以大于1)。

Theano允许通过下面3种方法来修改`THEANO_FLAG`变量的值。

第一种方法是直接修改系统的环境变量，如图2.6所示。



图2.6 设置THEANO\_FLAGS环境变量

第二种方法是在命令行中执行脚本时，添加THEANO\_FLAGS信息，但这种方法只适用于linux环境。

```
>>> THEANO_FLAGS='floatX=float32,device=gpu0' python ***.py
```

第三种方法是在代码文件中修改，但应该注意的是，这种方法比较容易写错，下面是常见的错误做法。

- 错误的做法。

```
>>> import theano
>>> THEANO_FLAGS='floatX=float64,device=gpu0'
```

首先来检查一下当前floatX的配置。

```
>>> print theano.config.floatX
float32
```

我们发现THEANO\_FLAG的配置信息没有生效，这是由于在导入Theano模块后，有些设置就不能被更改了，因此这时再设置环境变量并没有起作用。

- 正确的做法：应该在导入Theano模块之前，先设置环境变量，如下所示，结果也和我们预期的一样，floatX属性已经被正确设置为float64了。

```
>>> import os
>>> os.environ["THEANO_FLAGS"] = 'floatX=float64,device=cpu'
>>> import theano
>>> print theano.config.floatX
float64
```

## 2.7.2 通过.theanorc文件配置

.theanorc文件在操作系统的默认位置一般是：\$HOME/.theanorc。在Windows环境下，它的默认位置是：\$HOME/.theanorc或\$HOME/.theanorc.txt。

若存在下面的配置信息：

```
>>> THEANO_FLAGS='floatX=float32,device=cpu,lib.cnmem=1'
```

其对应的文件信息如下所示。

```
[global]
device=cpu
floatX=float32
```

```
[lib]
cnmem=1
```

在设置`.theanorc`文件时，如果属性直接位于`config`下，如`config.device`、`config.mode`等，这些属性应该放置在`global` section下；其他位于`config`子模块下的属性，例如`config.lib.cnmem`、`config.dnn.conv.algo_fwd`等，则应该放置在对应的子模块`section`。

## 2.8 常用的Debug技巧

Theano的调试功能是Theano的一个弱项，相比于Tensorflow等商业框架，Theano的调试比较麻烦，下面是一些常用的调试技巧。

- 打印详细的调试信息。通过在配置文件中设置`config.exception_verbosity`来输出详细的debug信息，有关配置和`config`模块可参考2.7节。
- 通过`eval`来查看或调试表达式的结果。对于`shared`变量，可以通过`value`或者`get_value`来查看变量的值，但对于其他`Tensorvariable`，符号变量是无法查看对应的值的，这时可以调用`eval`函数来查看。

```
>>> a = theano.shared(value = numpy.array([[1,2,3], [4,5,6]]))
>>> b = a.reshape(shape=(3,2))
>>> b.eval()
array([[1, 2],
       [3, 4],
       [5, 6]])
```

- 除了语法错误外，在编写深度学习项目的时候，遇到更多的是其他的逻辑上的错误，比如除0或矩阵相乘的维度不一致，例如一个大小为(5,10)与另一个大小为(20,10)的矩阵进行点乘操作，这些类型的错误往往只能在线上运行的时候发现。而Theano提供了`compute_test_value`帮助我们快速发现这一类型的错误，详细的使用方法可以参考`compute_test_value`的使用文档。

## 2.9 小结

本章简要讲解了Theano的一些常用基础语法点，但Theano库的设计和功​​能本身非常庞大，接口也很多，限于本书的篇幅，我们没有进行逐一探讨，如果读者想要了解更具体的功能实现，请参考Theano的官方文档，或到Google group的theano - users论坛寻求帮助。





---

# 第 2 部分

## 数学与机器学习基础篇

---

# 3

## 线性代数基础

线性代数是应用数学的一个重要分支，被广泛应用于自然科学和工程领域。线性代数，特别是矩阵运算也是很多机器学习算法，尤其是深度学习算法的基础。因此，本章将首先讲解线性代数相关的基础知识。

如果读者对本章的知识点比较熟悉，可以跳过这一章。在本章中，仅涉及与深度学习相关的线性代数知识点，因此，会忽略一些与机器学习无关的主题。如果读者想更深入全面地了解线性代数相关的知识，可以参考相关的文献[1,2]。

最后，为了方便起见，在本章的讨论中，如果没有特别的说明，讨论范围都是在实数域的范围。

### 3.1 标量、向量、矩阵和张量

在线性代数和机器学习中，数值计算通常由下面4种结构类型组成。

- 标量（scalar）：即一个数值，是计算的最小单元，通常用小写的拉丁字母来表示。
- 向量（vector）：向量是由多个标量数据构成的一维数组，如式(3.1)所示，通过下标索引来获取每一个元素，如 $\mathbf{x}_0$ 表示向量 $\mathbf{x}$ 的第一个元素（假设下标从0开始）。

$$\mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{n-1})^T \quad (3.1)$$

- 矩阵 (matrix)：矩阵是由标量数据构成的二维数组。在本书中，采用大写的拉丁字母来表示，如矩阵  $\mathbf{A} \in \mathbf{R}^{m \times n}$ 。与向量类似，可以通过给定行和列的下标索引来获取矩阵中的元素，如  $a_{0,0}$  表示矩阵中的第0行，第0列的元素。

$$\mathbf{A} = \begin{pmatrix} a_{0,0} & \cdots & a_{0,n-1} \\ \vdots & \ddots & \vdots \\ a_{m-1,0} & \cdots & a_{m-1,n-1} \end{pmatrix} \quad (3.2)$$

- 张量 (tensor)：在深度学习领域，很多时候，数据都是高于二维的，因此，需要一种能够表示任意维度的数据类型，这就是张量。如图3.1所示是一个三维的张量类型。

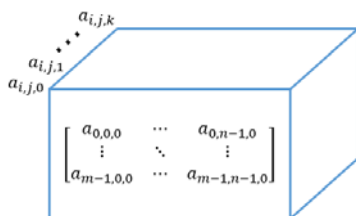


图 3.1 一个三维的张量类型

这4种类型对应Theano的创建方式如图3.2所示。

数据类型	Theano的创建方式
scalar	theano.tensor.scalar
vector	theano.tensor.vector
matrix	theano.tensor.matrix
tensor (3维)	theano.tensor.tensor3
tensor (4维)	theano.tensor.tensor4
tensor (大于4维)	theano.tensor.TensorType

图3.2 4种基本类型对应的Theano创建方式

## 3.2 矩阵初等变换

矩阵的初等变换是矩阵的一种非常重要的运算，它在求解线性方程组、逆矩阵中都起到了非常重要的作用。对于任意一个矩阵  $\mathbf{A} \in \mathbf{R}^{m \times n}$ ，把下面3种变换称为矩阵的初等行变换。

- 对调任意两行，记为  $r_i \leftrightarrow r_j$ 。

- 以不为零的数 $k$ 乘以某一行的所有元素，记为 $r_i \times k$ 。
- 把某一行的所有元素的 $k$ 倍加到另一行的对应的元素上去，记为 $r_i \times k + r_j$ 。

把上述的“行”换成“列”，可以得到等价的初等列变换的定义，初等行变换与初等列变换统称为初等变换。

**矩阵等价：**如果矩阵 $A$ 经过有限次初等变换变为矩阵 $B$ ，那么称矩阵 $A$ 与矩阵 $B$ 等价，记为 $A \sim B$ 。

矩阵之间的等价关系有下面的性质。

- 自反性： $A \sim A$ 。
- 对称性：若 $A \sim B$ ，则 $B \sim A$ 。
- 传递性：若 $A \sim B$ ， $B \sim C$ ，则 $A \sim C$ 。

### 3.3 线性相关与向量空间

在3.1节，介绍了向量的定义，由多个同维度的列向量构成的集合称为向量组，矩阵可以看成是由行向量或者列向量构成的向量组。本节将讲解向量组的几个重要概念，包括线性组合、线性相关以及最大线性无关组，最终得到矩阵的秩概念。

#### 1. 线性组合

给定向量组 $A: \mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n$  ( $\mathbf{a}_i \in \mathbf{R}^m$ )，对于任意一组实数 $k_1, k_2, \dots, k_n$ ，把表达式：

$$k_1 \mathbf{a}_1 + k_2 \mathbf{a}_2 + \dots + k_n \mathbf{a}_n \quad (3.3)$$

称为向量组 $A$ 的一个线性组合， $k_1, k_2, \dots, k_n$ 称为向量组的系数。

对于任意一个 $m$ 维向量 $\mathbf{b}$ ，如果存在一组实数 $k_1, k_2, \dots, k_n$ ，使得：

$$\mathbf{b} = k_1 \mathbf{a}_1 + k_2 \mathbf{a}_2 + \dots + k_n \mathbf{a}_n \quad (3.4)$$

成立，则称向量 $\mathbf{b}$ 可以被向量组 $A: \mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n$ 线性表示。

对于任意的实数集 $\{k_1, k_2, \dots, k_n\}$ ，我们把由式(3.3)构成的所有向量集合，称为向量空间 $\{k_1 \mathbf{a}_1 + k_2 \mathbf{a}_2 + \dots + k_n \mathbf{a}_n, k_i \in \mathbf{R}\}$ 。

比如，由向量组 $\{(1,0,0), (0,1,0), (0,0,1)\}$ 和任意的一组实数 $\{k_1, k_2, k_3\}$ 构成了全局的三维空间。

## 2. 线性相关

给定向量组 $\mathbf{A}: \mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n$ ，如果存在不全为0的实数 $k_1, k_2, \dots, k_n$ ，使得：

$$k_1 \mathbf{a}_1 + k_2 \mathbf{a}_2 + \dots + k_n \mathbf{a}_n = \mathbf{0} \quad (3.5)$$

成立，则称向量组 $\mathbf{A}$ 是线性相关的。反之，称向量组 $\mathbf{A}$ 是线性无关的。

## 3. 向量组的秩

设有向量组 $\mathbf{A}: \mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n$ ，如果能从中选出由 $r$ 个子向量构成的子向量组 $\mathbf{A}_0: \mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_r$ ， $r < n$ ，满足：

- $\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_r$ 线性无关。
- 向量组 $\mathbf{A}$ 的任意 $(r+1)$ 个向量构成的子向量组都是线性相关的。

那么，称子向量组 $\mathbf{A}_0: \mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_r$ 是向量组 $\mathbf{A}$ 的一个最大线性无关向量组，最大线性无关向量组包含的向量个数 $r$ 称为向量组 $\mathbf{A}$ 的秩。

前面提到向量组和矩阵是一种等价关系，事实上，如果可以把矩阵看成是由所有行向量构成行向量组，这样矩阵的行秩就等于行向量组的秩，同理，矩阵的列秩就等于列向量组的秩。

设矩阵 $\mathbf{A}$ 的行秩为 $R(\mathbf{A})_r$ ，列秩为 $R(\mathbf{A})_c$ ，则有 $R(\mathbf{A})_r = R(\mathbf{A})_c$ ，因此，把矩阵 $\mathbf{A}$ 的行秩与列秩统称为矩阵 $\mathbf{A}$ 的秩。

## 3.4 范数

范数（Norm）是数学中的一种基本概念，在泛函分析中，范数是一种定义在赋范线性空间中的函数，满足相应条件后的函数都可以被称为范数，本节考察向量范数和矩阵范数的定义，并给出常用的几种范数计算公式。

### 3.4.1 向量范数

在泛函分析中，向量范数是衡量向量大小的一种度量方式。在形式上，向量范数是一个定义域为任意线性空间向量的函数，它把一个向量 $\mathbf{v}$ 映射为一个非负实数值 $\mathbf{R}$ ，即满足 $f: \mathbf{v} \rightarrow \mathbf{R}$ 。

从几何角度来说，向量 $\mathbf{x}$ 的范数是度量从原点 $O$ 到点 $\mathbf{x}$ 的距离。从广义角度来说，对于任意一个函数 $f: \mathbf{v} \rightarrow \mathbf{R}$ ， $f$ 只要同时满足下面3个条件就可以称之为范数。

- 非负性： $f(\mathbf{x}) \leq 0$ ，且当 $f(\mathbf{x}) = 0$ 时，必有 $\mathbf{x} = 0$ 成立。
- 三角不等式性： $f(\mathbf{x} + \mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y})$ 。
- 齐次性： $\forall \mathbf{x} \in \mathbf{R}^n$ ， $f(a\mathbf{x}) = |a| \times f(\mathbf{x})$ 。

在机器学习中，特别是在模型最优化时，范数是正则化的主要手段，用来衡量模型的复杂度，其中，最常用到的是 $p$ 范数， $p \in \mathbf{R}$ ， $p \geq 1$ 。

$p$ 范数的定义如下。

$$\|\mathbf{x}\|_p = \left( \sum_i |x_i|^p \right)^{\frac{1}{p}} \quad (3.6)$$

容易验证 $\|\mathbf{x}\|_p$ 满足范数成立的三个条件，读者可自行推导，这里不再详述。下面分析几个常用的 $p$ -范数，为了更形象地理解不同 $p$ 范数之间的区别，将分别给出在二维向量空间中，不同 $p$ 值对应的单位范数，即 $\|\mathbf{x}\|_p = 1$ 的图形表示。

**0范数：**这是一个特殊的 $p$ -范数，它表示向量中非0元素的个数。注意，0-范数不是通过式(3.6)来定义。

事实上，我们会发现0范数不是真正的范数，因为它并不满足前面提到的齐次性性质，也就是 $f(a\mathbf{x}) \neq |a| \times f(\mathbf{x})$ ，这是因为一个向量乘上一个实数 $a$ ，若 $a \neq 0$ ，那么该操作并不会改变向量中非0元素的个数。

**1范数：**也被称为绝对值范数，大小等于向量的每个元素绝对值之和，即：

$$\|\mathbf{x}\|_1 = \sum_i |x_i| \quad (3.7)$$

其中，单位范数即满足 $\|\mathbf{x}\|_1 = 1$ 的点 $\{(x_1, x_2)\}$ ，如图3.3所示。

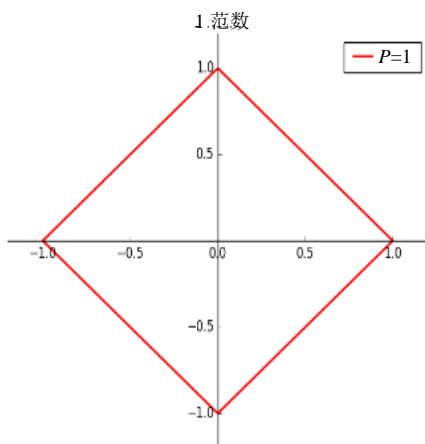


图3.3 1范数单位圆

**2范数:**也被称为欧几里得范数,它的大小表示从原点到当前点的欧几里得距离,即:

$$\|\mathbf{x}\|_2 = \sqrt{|x_1|^2 + |x_2|^2 + \cdots + |x_n|^2} \quad (3.8)$$

2范数也是通常意义上的模,如图3.4所示。

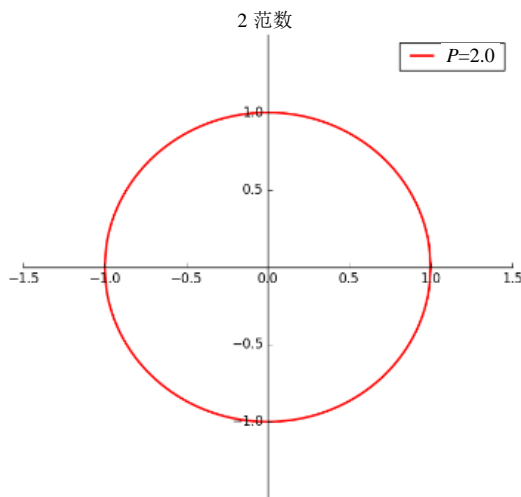


图3.4 2范数单位圆

**$\infty$ 范数:**也被称为最大范数,它的值等于向量中每个元素的绝对值的最大值。也就是 $\|\mathbf{x}\|_p = \max_i(|x_i|)$ ,下面给出如何由式(3.6)得到 $\infty$ 范数的表达式。

证明: 设向量 $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$ ,  $x_k = \max_i(|x_i|)$ , 有

$$\begin{aligned}
\lim_{p \rightarrow \infty} \|\mathbf{x}\|_p &= \lim_{p \rightarrow \infty} (x_1^p + x_2^p + \cdots + x_n^p)^{\frac{1}{p}} \\
&= \lim_{p \rightarrow \infty} \left( \left( \frac{x_1}{x_k} \right)^p \times x_k^p + \left( \frac{x_2}{x_k} \right)^p \times x_k^p + \cdots + \left( \frac{x_n}{x_k} \right)^p \times x_k^p \right)^{\frac{1}{p}} \\
&= x_k \times \lim_{p \rightarrow \infty} \left( \left( \frac{x_1}{x_k} \right)^p + \left( \frac{x_2}{x_k} \right)^p + \cdots + \left( \frac{x_n}{x_k} \right)^p \right)^{\frac{1}{p}} \quad (3.9)
\end{aligned}$$

其中：

$$\lim_{p \rightarrow \infty} \left( \left( \frac{x_1}{x_k} \right)^p + \left( \frac{x_2}{x_k} \right)^p + \cdots + \left( \frac{x_n}{x_k} \right)^p \right)^{\frac{1}{p}} \leq \lim_{p \rightarrow \infty} \left( \left( \frac{x_k}{x_k} \right)^p + \left( \frac{x_k}{x_k} \right)^p + \cdots + \left( \frac{x_k}{x_k} \right)^p \right)^{\frac{1}{p}} = \lim_{p \rightarrow \infty} (n)^{\frac{1}{p}}$$

即：

$$\lim_{p \rightarrow \infty} \left( \left( \frac{x_1}{x_k} \right)^p + \left( \frac{x_2}{x_k} \right)^p + \cdots + \left( \frac{x_n}{x_k} \right)^p \right)^{\frac{1}{p}} \leq \lim_{p \rightarrow \infty} (n)^{\frac{1}{p}} = 1 \quad (3.10)$$

又因为：

$$\lim_{p \rightarrow \infty} \left( \left( \frac{x_1}{x_k} \right)^p + \left( \frac{x_2}{x_k} \right)^p + \cdots + \left( \frac{x_n}{x_k} \right)^p \right)^{\frac{1}{p}} \geq \lim_{p \rightarrow \infty} \left( \left( \frac{x_k}{x_k} \right)^p \right)^{\frac{1}{p}} = 1 \quad (3.11)$$

把式(3.10)与式(3.11)代入式(3.9)，得：

$$x_k \leq \lim_{p \rightarrow \infty} \|\mathbf{x}\|_p \leq x_k \quad (3.12)$$

因此，由夹逼准则<sup>[4,5]</sup>可得 $\lim_{p \rightarrow \infty} \|\mathbf{x}\|_p = \max_i(|x_i|)$ 成立，如图 3.5 所示。

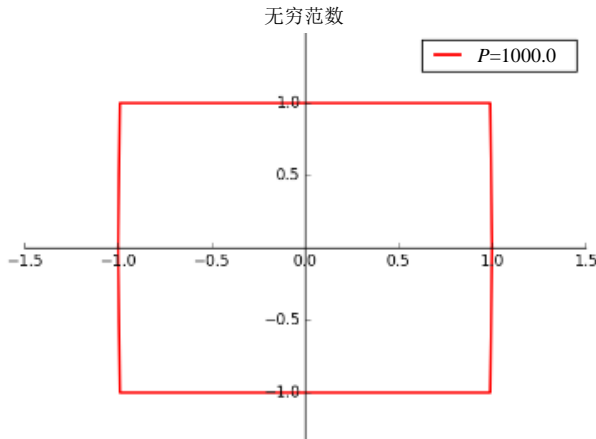


图3.5 无穷范数单位圆



总结一下, 对于一个二维向量  $\mathbf{x} = (x_1, x_2)^T$ , 它的单位范数随  $p$  值增加的变化过程, 如图3.6所示。

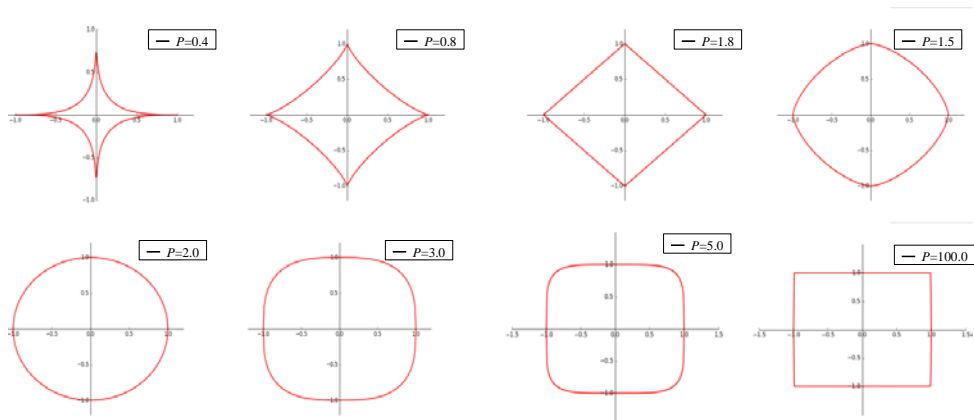


图3.6 不同  $p$  值对应的单位范数

### 3.4.2 矩阵范数

前面讨论了常用的向量范数, 范数的概念同样可以推广到矩阵领域, 由此产生了矩阵范数的概念: 对于任意一个函数  $f: \mathbf{A} \rightarrow \mathbf{R}$ , 函数  $f$  只要同时满足下面4个条件就可以称之为矩阵范数。

- 非负性:  $f(\mathbf{A}) \geq 0$ , 且当  $f(\mathbf{A}) = 0$  时, 必要  $\mathbf{A} = 0$  成立。
- 齐次性:  $\forall \mathbf{A} \in \mathbf{R}^{m \times n}$ ,  $f(a\mathbf{A}) = |a| \times f(\mathbf{A})$ 。
- 三角不等式性:  $f(\mathbf{A} + \mathbf{B}) \leq f(\mathbf{A}) + f(\mathbf{B})$ 。
- 矩阵乘法的相容性: 对于任意两个可乘的矩阵  $\mathbf{A} \in \mathbf{R}^{m \times k}$  和矩阵  $\mathbf{B} \in \mathbf{R}^{k \times n}$ , 满足  $f(\mathbf{AB}) \leq f(\mathbf{A}) \times f(\mathbf{B})$ 。

为了与向量范数对应, 用  $\|\mathbf{A}\|_p$  来表示矩阵的  $p$  范数, 矩阵范数同样有多种不同的形式, 但最常用的是与向量范数紧密相连的诱导范数, 首先给出诱导范数的概念。

定义1: 设  $\|\mathbf{x}\|_a$  是向量  $\mathbf{x}$  的范数,  $\|\mathbf{A}\|_b$  是矩阵  $\mathbf{A}$  的范数, 如果对于任意的矩阵  $\mathbf{A}$  与向量  $\mathbf{x}$  满足:

$$\|\mathbf{Ax}\|_a \leq \|\mathbf{A}\|_b \times \|\mathbf{x}\|_a \quad (3.13)$$

成立, 则称矩阵范数  $\|\mathbf{A}\|_b$  与向量范数  $\|\mathbf{x}\|_a$  是相容的。

定义2：设 $\|\mathbf{x}\|_p$ 是向量 $p$ 范数，定义：

$$\|\mathbf{A}\|_p = \max_{\mathbf{x} \neq 0} \frac{\|\mathbf{Ax}\|_p}{\|\mathbf{x}\|_p} = \max_{\|\mathbf{x}\|_p=1} \|\mathbf{Ax}\|_p \quad (3.14)$$

则 $\|\mathbf{A}\|_p$ 是一个矩阵范数，并且称该矩阵范数是由向量范数 $\|\mathbf{x}\|_p$ 所诱导的诱导范数。

下面验证式(3.14)定义的矩阵范数满足4个矩阵范数的条件。

- 满足非负性。

对于任意的向量 $\mathbf{x} \in \mathbf{R}^n$ ， $\|\mathbf{x}\|_p = 1$ ，由于 $\|\mathbf{Ax}\|_p \geq 0$ （向量范数具有非负性），故

$$\|\mathbf{A}\|_p = \max_{\|\mathbf{x}\|_p=1} \|\mathbf{Ax}\|_p \geq 0 \quad (3.15)$$

进一步由向量范数的性质，有：

$$\begin{aligned} \|\mathbf{A}\|_p &= \max_{\|\mathbf{x}\|_p=1} \|\mathbf{Ax}\|_p = 0 \\ \Leftrightarrow \|\mathbf{Ax}\|_p &= 0, \forall \mathbf{x} \in \mathbf{R}^n, \|\mathbf{x}\|_p = 1 \\ \Leftrightarrow \mathbf{Ax} &= 0, \forall \mathbf{x} \in \mathbf{R}^n, \|\mathbf{x}\|_p = 1 \\ \Leftrightarrow \mathbf{A} &= 0 \end{aligned}$$

故非负性成立。

- 满足齐次性。

$$\begin{aligned} \|\mathbf{aA}\|_p &= \max_{\|\mathbf{x}\|_p=1} \|(\mathbf{aA})\mathbf{x}\|_p \\ &= \max_{\|\mathbf{x}\|_p=1} \|\mathbf{a}(\mathbf{Ax})\|_p \\ &= |\mathbf{a}| \max_{\|\mathbf{x}\|_p=1} \|(\mathbf{Ax})\|_p = |\mathbf{a}| \|\mathbf{A}\|_p \end{aligned}$$

- 满足三角不等式性。

$$\|\mathbf{A} + \mathbf{B}\|_p = \max_{\|\mathbf{x}\|_p=1} \|(\mathbf{A} + \mathbf{B})\mathbf{x}\|_p = \max_{\|\mathbf{x}\|_p=1} \|(\mathbf{Ax} + \mathbf{Bx})\|_p \quad (3.16)$$

利用向量范数的三角不等式，可进一步化简式(3.16)：

$$\begin{aligned} \|\mathbf{A} + \mathbf{B}\|_p &= \max_{\|\mathbf{x}\|_p=1} \|(\mathbf{A} + \mathbf{B})\mathbf{x}\|_p = \max_{\|\mathbf{x}\|_p=1} \|(\mathbf{Ax} + \mathbf{Bx})\|_p \\ &\leq \max_{\|\mathbf{x}\|_p=1} (\|\mathbf{Ax}\|_p + \|\mathbf{Bx}\|_p) \end{aligned}$$

$$\leq \max_{\|x\|_p=1} \|Ax\|_p + \max_{\|x\|_p=1} \|Bx\|_p = \|A\|_p + \|B\|_p$$

• 满足矩阵乘法的相容性。

$$\|AB\|_p = \max_{\|x\|_p=1} \|(AB)x\|_p = \max_{\|x\|_p=1} \|A(Bx)\|_p \quad (3.17)$$

由式(3.14), 有:

$$\|A\|_p = \max_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} \geq \frac{\|Ax\|_p}{\|x\|_p} \Leftrightarrow \|A\|_p \|x\|_p \geq \|Ax\|_p \quad (3.18)$$

把式(3.18)代入式(3.17)可得:

$$\begin{aligned} \|AB\|_p &= \max_{\|x\|_p=1} \|(AB)x\|_p = \max_{\|x\|_p=1} \|A(Bx)\|_p \\ &\leq \max_{\|x\|_p=1} (\|A\|_p \times \|Bx\|_p) = \|A\|_p \times \max_{\|x\|_p=1} \|Bx\|_p = \|A\|_p \times \|B\|_p \end{aligned}$$

由向量 $p$ 范数, 可以诱导出常用的矩阵 $p$ 范数。

**1范数:** 记为 $\|A\|_1$ , 是由向量1范数诱导的矩阵范数, 矩阵1范数的定义为:

$$\|A\|_1 = \max_j \left( \sum_{i=1}^m |a_{ij}| \right), \quad j = 1, 2, \dots, n \quad (3.19)$$

矩阵1范数也被称为列和范数。

**2范数:** 记为 $\|A\|_2$ , 是由向量2范数诱导的矩阵范数, 矩阵2范数的定义为:

$$\|A\|_2 = \max_j \sqrt{(\lambda_j(A^T A))} \quad (3.20)$$

$\lambda_j(A^T A)$ 表示矩阵 $A^T A$ 的第 $j$ 个特征值, 2范数也被称为谱范数。

**$\infty$ 范数:** 记为 $\|A\|_\infty$ , 是由向量 $\infty$ 范数诱导的矩阵范数, 矩阵 $\infty$ 范数的定义为:

$$\|A\|_\infty = \max_i \left( \sum_{j=1}^n |a_{ij}| \right), \quad i = 1, 2, \dots, m \quad (3.21)$$

矩阵 $\infty$ 范数也被称为行和范数。

此外, 在深度学习中, 还经常用到另外一个矩阵范数, 称为**Frobenious范数**, 简称**F范数**, 其定义为:

$$\|\mathbf{A}\|_F = \left( \sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2 \right)^{1/2} \quad (3.22)$$

在后面章节中，很多模型的最优化问题，如 PCA（3.9节）和自编码器（第10章）等，都能转化为F范数的最优化问题。

### 3.5 特殊的矩阵与向量

本节首先介绍几种特殊的矩阵与向量，一般来说，对于任意一个矩阵，都可以化为这些特殊矩阵的组合形式。

（1）对角矩阵：记为 $\Sigma$ ，对角矩阵除了对角线元素之外，其余元素均为0。形式化地说，对于任意的一个矩阵 $\mathbf{A} \in \mathbf{R}^{m \times n}$ ，矩阵 $\mathbf{A}$ 是对角矩阵当且仅当 $\mathbf{A}_{i,j} = 0$ ，其中 $i \neq j$ 。要特别注意的是，对角矩阵不一定是方阵，则 $m$ 可以不等于 $n$ 。

$$\mathbf{A} = \begin{pmatrix} 1 & 0 \\ 0 & 2 \\ 0 & 0 \end{pmatrix} \quad (3.23)$$

对于对角矩阵是 $n$ 阶方阵，那么可以把对角矩阵记为：

$$\Sigma = \text{diag}(\mathbf{v}) = \text{diag}([\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n]^T) \quad (3.24)$$

其中 $\mathbf{v} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n]^T$ 是由对角线元素组成的向量。对角矩阵具有很多优秀的性质，在矩阵运算中的效率也非常高。

假设现有向量 $\mathbf{v} = [\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n]^T$ ， $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]^T$ ，则满足：

$$\text{diag}(\mathbf{v}) \times \mathbf{x} = \mathbf{v} \times \mathbf{x} \quad (3.25)$$

$$\text{diag}(\mathbf{v})^{-1} = \text{diag}([1/\mathbf{v}_1, 1/\mathbf{v}_2, \dots, 1/\mathbf{v}_n]^T) \quad (3.26)$$

（2）对称矩阵：对于任意一个 $n$ 阶方阵 $\mathbf{A}$ ，若 $\mathbf{A}$ 满足：

$$\mathbf{A} = \mathbf{A}^T \quad (3.27)$$

成立，则称方阵 $\mathbf{A}$ 是一个对称矩阵。

（3）单位向量：对于任意给定的向量 $\mathbf{v}$ ，如果 $\mathbf{v}$ 的欧几里得范数为1，即满足：

$$\|\mathbf{x}\|_2 = 1 \quad (3.28)$$

成立，则称向量 $\boldsymbol{v}$ 为单位向量。

(4) 正交向量：假设现有向量 $\boldsymbol{v} = [v_1, v_2, \dots, v_n]^T$ ， $\boldsymbol{x} = [x_1, x_2, \dots, x_n]^T$ ，若满足：

$$\boldsymbol{v}^T \odot \boldsymbol{x} = 0 \quad (3.29)$$

成立，其中 $\odot$ 表示向量的点积运算，则称向量 $\boldsymbol{v}$ 与向量 $\boldsymbol{x}$ 正交。

(5) 正交矩阵：对于任意一个 $n$ 阶方阵 $\boldsymbol{A}$ ，若矩阵的行向量之间相互正交，且行向量均为单位向量，也就是满足：

$$\boldsymbol{A}\boldsymbol{A}^T = \boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I} \quad (3.30)$$

成立，则称方阵 $\boldsymbol{A}$ 是一个正交矩阵，从上式也可以看出，若 $\boldsymbol{A}$ 是一个正交矩阵，那么必有 $\boldsymbol{A}^T = \boldsymbol{A}^{-1}$ 成立。

## 3.6 特征值分解

分而治之是算法设计的一种常用策略，通过把一个物体分解为不同的子部分，有时可以更好地理解整体特性。

以整数为例，由算术基本定理<sup>[3]</sup>可知，对于任意一个大于1的自然数 $\boldsymbol{N}$ ， $\boldsymbol{N}$ 都可以分解为有限个素数的乘积，即满足 $\boldsymbol{N} = p_1^{a_1} \times p_2^{a_2} \times \dots \times p_n^{a_n}$ ，成立，其中 $p_i$ 为素数，且 $p_1 < p_2 < \dots < p_n$ ， $a_i$ 为正整数。

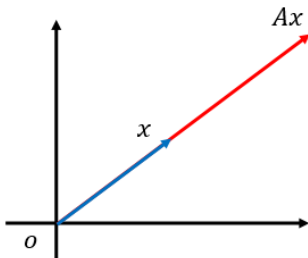
对任意一个整数，通过素数唯一性分解，可以得到很多有用的性质，例如，30可以被分解为 $30 = 2 \times 3 \times 5$ 。通过这个表示，可以知道30不能被7整除。由约数个数定理可知，30一共有8个约数，由约数和定理可知，它的所有约数之和为 $(1+2) \times (1+3) \times (1+5) = 72$ 。

同理，对于矩阵来说，也可以把一个矩阵分解为几个不同的子矩阵，通过观察每个子矩阵可以得到矩阵本身的有用性质。本节介绍一个常用的矩阵分解方法——特征值分解：将矩阵分解为其特征值和特征向量表示的矩阵之积的方法。注意，本节讨论的矩阵都是 $n$ 阶方阵。

**特征值与特征向量：**设 $\boldsymbol{A}$ 是 $n$ 阶方阵，如果存在实数 $\lambda$ 和 $n$ 维的非0向量 $\boldsymbol{x}$ ，满足：

$$\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x} \quad (3.31)$$

成立，那么把数 $\lambda$ 称为方阵 $\mathbf{A}$ 的特征值，非0向量 $\mathbf{x}$ 称为方阵 $\mathbf{A}$ 对应于特征值 $\lambda$ 的特征向量。



从几何角度来看，特征向量的方向经过线性变换后，保持在同一直线上，这时或者方向不变( $\lambda > 0$ )，或者方向相反( $\lambda < 0$ )，或者线性变换为0向量( $\lambda = 0$ )。

现假设方阵 $\mathbf{A}$ 有 $n$ 个线性无关的特征向量 $\{\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^n\}$ ，它们对应的特征值分别为 $\{\lambda_1, \lambda_2, \dots, \lambda_n\}$ 。

把 $n$ 个线性无关的特征向量组合起来构成一个新方阵，每一列是一个特征向量：

$$\mathbf{V} = [\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^n] \quad (3.32)$$

用同样的方式，用特征值构成一个新的矩阵，新矩阵是一个对角矩阵，对角线上的元素就是特征值：

$$\mathbf{\Sigma} = \text{diag}([\lambda_1, \lambda_2, \dots, \lambda_n]^T) \quad (3.33)$$

那么，方阵 $\mathbf{A}$ 的特征值分解可以表示为：

$$\mathbf{A} = \mathbf{V} \times \mathbf{\Sigma} \times \mathbf{V}^{-1} \quad (3.34)$$

这样就完成了一个方阵的特征值分解，但要注意的是**并不是所有的方阵都存在特征值分解**，一个方阵 $\mathbf{A} \in \mathbf{R}^{n \times n}$ 能够进行特征分解的充要条件是 $\mathbf{A}$ 含有 $n$ 个线性无关的特征向量。

### 3.7 奇异值分解

奇异值分解 (Singular Value Decomposition, 简称SVD)，是线性代数中另一种非常重要的矩阵分解算法，3.6节讲解了矩阵的特征分解，但特征值分解只适用于 $n$ 阶方阵，并且不是所有的 $n$ 阶方阵都存在特征分解，而SVD的应用更广，它适用于任意给

定的 $m \times n$ 阶实数矩阵 $\mathbf{A}$ 。除了适用于降维，SVD还能应用于很多机器学习的工程领域，如推荐系统<sup>[7]</sup>、文本主题相关性等。图3.7展示了SVD的分解样例。

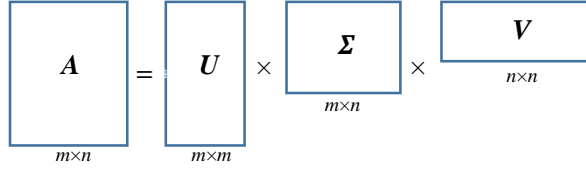


图3.7 SVD分解样例

形式化来说，设矩阵 $\mathbf{A} \in \mathbf{R}^{m \times n}$ 、 $\mathbf{U} \in \mathbf{R}^{m \times m}$ 、 $\mathbf{\Sigma} \in \mathbf{R}^{m \times n}$ 、 $\mathbf{V} \in \mathbf{R}^{n \times n}$ ，则矩阵 $\mathbf{A}$ 可以分解为下面的形式：

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \quad (3.35)$$

其中， $\mathbf{U} = \{\mathbf{u}^1, \mathbf{u}^2, \dots, \mathbf{u}^m\}$ 是一个 $m$ 阶方阵， $\mathbf{u}^i$ 的值是矩阵 $\mathbf{A}\mathbf{A}^T$ 的第 $i$ 大的特征值对应的特征向量。 $\mathbf{u}^i$ 也称为矩阵 $\mathbf{A}$ 的左奇异向量。

$\mathbf{V} = \{\mathbf{v}^1, \mathbf{v}^2, \dots, \mathbf{v}^n\}$ 是一个 $n$ 阶方阵，其中 $\mathbf{v}^i$ 的值是矩阵 $\mathbf{A}^T\mathbf{A}$ 的第 $i$ 大的特征值对应的特征向量。 $\mathbf{v}^i$ 也被称为矩阵 $\mathbf{A}$ 的右奇异向量。

$\mathbf{\Sigma}$ 是一个对角矩阵，但要注意的是 $\mathbf{\Sigma}$ 不一定是方阵， $\mathbf{\Sigma} \in \mathbf{R}^{m \times n}$ ，设其对角线元素为 $(\sigma_1, \sigma_2, \dots, \sigma_k)$ ，其中 $\sigma_i$ 是矩阵 $\mathbf{A}^T\mathbf{A}$ 的第 $i$ 大的特征值的平方根，记为 $\sigma_i = \sqrt{\lambda_i(\mathbf{A}^T\mathbf{A})}$ 。

在很多情况下，不需要使分解的结果 $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ 准确还原矩阵 $\mathbf{A}$ ，只需要接近于 $\mathbf{A}$ 即可。事实上，奇异值 $\sigma$ 与特征值的性质相似，在矩阵 $\mathbf{\Sigma}$ 中也是从大到小排列，而且 $\sigma$ 的值减少特别快，在很多情况下，前10%甚至1%的奇异值的和就占了全部的奇异值之和的99%以上了。也就是说，可以用前 $r$ 大的奇异值来近似描述矩阵，因此可以将式(3.33)化为：

$$\mathbf{A} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \quad (3.36)$$

其中， $\mathbf{U} \in \mathbf{R}^{m \times r}$ 、 $\mathbf{\Sigma} \in \mathbf{R}^{r \times r}$ 、 $\mathbf{V} \in \mathbf{R}^{r \times n}$ ，一般情况下 $r$ 的值要远远小于 $m$ 值和 $n$ 值，因此 $m \times r + r \times r + r \times n$ 要远远小于 $m \times n$ 。我们只需要保存 $\mathbf{U}$ 、 $\mathbf{\Sigma}$ 和 $\mathbf{V}$ 这3个小矩阵就能还原矩阵 $\mathbf{A}$ 。

验证本节开头提到的，为什么SVD可以适用于任意的矩阵。3.6节提到矩阵的特征值分解只适用于方阵，且要求方阵含有 $n$ 个线性无关的特征向量。而对于SVD，它的奇异向量与奇异值均是通过求取矩阵 $\mathbf{A}^T\mathbf{A}$ 或矩阵 $\mathbf{A}\mathbf{A}^T$ 的特征值和特征向量得到，由于

$\mathbf{A}\mathbf{A}^T$ 和 $\mathbf{A}^T\mathbf{A}$ 均为对称矩阵，故必有 $n$ 个的特征向量都线性无关，因此，对于任意的矩阵 $\mathbf{A}$ ，它的奇异值分解必存在。

### 3.8 迹运算

一个 $n$ 阶方阵 $\mathbf{A}$ 的主对角线上各个元素的总和被称为矩阵 $\mathbf{A}$ 的迹，一般记为 $\text{tr}\mathbf{A}$ 。则有：

$$\text{tr}\mathbf{A} = \sum_i \mathbf{A}_{i,i} \quad (3.37)$$

矩阵的迹有很多重要的性质。

- 矩阵 $\mathbf{A}$ 的迹与矩阵 $\mathbf{A}$ 的F范数有紧密的联系，满足

$$\|\mathbf{A}\|_F = \sqrt{\text{tr}(\mathbf{A}\mathbf{A}^T)} \quad (3.38)$$

- 矩阵 $\mathbf{A}$ 的迹与矩阵 $\mathbf{A}$ 的特征值有紧密的联系，设 $(\lambda_1, \lambda_2, \dots, \lambda_n)$ 为矩阵 $\mathbf{A}$ 的所有特征值，那么满足：

$$\text{tr}\mathbf{A} = \sum_i \mathbf{A}_{i,i} = \sum_i \lambda_i \quad (3.39)$$

即矩阵的迹是所有特征值之和。

- 矩阵的迹满足下面几个等价关系。

$$\text{tr}\mathbf{A} = \text{tr}\mathbf{A}^T \quad (3.40)$$

$$\text{tr}(m\mathbf{A} + n\mathbf{B}) = m \times \text{tr}\mathbf{A} + n \times \text{tr}\mathbf{B} \quad (3.41)$$

$$\text{tr}(\mathbf{ABC}) = \text{tr}(\mathbf{CAB}) = \text{tr}(\mathbf{BCA}) \quad (3.42)$$

其中可以把式(3.42)推广到 $n$ 个矩阵相乘：

$$\text{tr}\left(\prod_{i=1}^n \mathbf{A}_i\right) = \text{tr}\left(\mathbf{A}_n \prod_{i=1}^{n-1} \mathbf{A}_i\right) \quad (3.43)$$

- 对矩阵的迹运算进行求导具有下面的重要性质。我们利用解析法求解线性模型的最优参数时，将用到这些性质（详细请参见第6章式(6.10)的推导过程）。

$$\nabla_{\mathbf{A}} \text{tr}(\mathbf{AB}) = \mathbf{B}^T \quad (3.44)$$



$$\nabla_{\mathbf{A}^T} f(\mathbf{A}) = (\nabla_{\mathbf{A}} f(\mathbf{A}))^T \quad (3.45)$$

$$\nabla_{\mathbf{A}} \text{tr}(\mathbf{A}\mathbf{B}\mathbf{A}^T \mathbf{C}) = \mathbf{C}\mathbf{A}\mathbf{B} + \mathbf{C}^T \mathbf{A}\mathbf{B}^T \quad (3.46)$$

$$\nabla_{\mathbf{A}^T} \text{tr}(\mathbf{A}\mathbf{B}\mathbf{A}^T \mathbf{C}) = \mathbf{B}^T \mathbf{A}^T \mathbf{C}^T + \mathbf{B}\mathbf{A}^T \mathbf{C} \quad (3.47)$$

### 3.9 样例：主成分分析

主成分分析 (Principal Components Analysis, 简称PCA), 是机器学习中一种常用的降维算法, 也是与线性代数紧密联系的一个算法。

考察这样的一个问题, 设在 $n$ 维空间中有 $m$ 个样本点:  $\{x^1, x^2, \dots, x^m\}$ , 现在对这些点进行压缩, 使其投影到 $k$ 维空间中, 其中 $k < n$ , 使其损失的信息最小。

设投影到 $k$ 维空间后的新坐标系为 $\{\mathbf{w}^1, \mathbf{w}^2, \dots, \mathbf{w}^k\}$ , 其中 $\mathbf{w}^i \in \mathbf{R}^n$ ,  $\mathbf{w}^i$ 是标准正交基向量, 即满足:

$$\|\mathbf{w}^i\|_2 = 1, \quad \mathbf{w}^{iT} \mathbf{w}^j = 0 \quad (i \neq j)$$

设矩阵 $\mathbf{W} = \{\mathbf{w}^1, \mathbf{w}^2, \dots, \mathbf{w}^k\}$ , 是一个大小为 $n \times k$ 维的正交矩阵, 也是投影矩阵,  $\mathbf{X} = \{x^1, x^2, \dots, x^m\}$ 是训练数据集, 是一个大小为 $n \times m$ 维的矩阵, 由矩阵乘法的定义可知, 投影到 $k$ 维空间的点的坐标为 $\mathbf{Z} = \mathbf{W}^T \mathbf{X}$ 。

利用该坐标系重构数据, 即把数据集 $\mathbf{Z}$ 从 $k$ 维空间重新映射回 $n$ 维空间, 得到新的坐标点为 $\mathbf{X}^* = \mathbf{W}\mathbf{Z} = \mathbf{W}\mathbf{W}^T \mathbf{X}$ 。

一种合理的设想是重构后的点 $\mathbf{X}^*$ 与原始的数据点之间距离最小, 即PCA可转化为求解约束最优化问题:

$$\begin{aligned} \min_{\mathbf{W}} \|\mathbf{X} - \mathbf{X}^*\|_F^2 &= \min_{\mathbf{W}} \|\mathbf{X} - \mathbf{W}\mathbf{W}^T \mathbf{X}\|_F^2 \\ \text{s.t. } &\mathbf{W}^T \mathbf{W} = \mathbf{I} \end{aligned} \quad (3.48)$$

由F范数与矩阵迹的关系式(3.38), 进一步化简式(3.48),

$$\begin{aligned} \min_{\mathbf{W}} \|\mathbf{X} - \mathbf{W}\mathbf{W}^T \mathbf{X}\|_F^2 &= \min_{\mathbf{W}} \text{tr}((\mathbf{X} - \mathbf{W}\mathbf{W}^T \mathbf{X})^T (\mathbf{X} - \mathbf{W}\mathbf{W}^T \mathbf{X})) \\ &= \min_{\mathbf{W}} \text{tr}(\mathbf{X}^T \mathbf{X} - \mathbf{X}^T \mathbf{W}\mathbf{W}^T \mathbf{X} - \mathbf{X}^T \mathbf{W}\mathbf{W}^T \mathbf{X} + \mathbf{X}^T \mathbf{W}\mathbf{W}^T \mathbf{W}\mathbf{W}^T \mathbf{X}) \end{aligned} \quad (3.49)$$

与 $\mathbf{W}$ 无关的项并不会影响最优化的结果, 并且由约束条件 $\mathbf{W}^T \mathbf{W} = \mathbf{I}$ , 进一步化简式(3.49), 可得:

$$\begin{aligned}
\min_{\mathbf{W}} \|\mathbf{X} - \mathbf{W}\mathbf{W}^T\mathbf{X}\|_F^2 &= \min_{\mathbf{W}} \text{tr}((\mathbf{X} - \mathbf{W}\mathbf{W}^T\mathbf{X})^T(\mathbf{X} - \mathbf{W}\mathbf{W}^T\mathbf{X})) \\
&= \min_{\mathbf{W}} \text{tr}(\mathbf{X}^T\mathbf{X} - \mathbf{X}^T\mathbf{W}\mathbf{W}^T\mathbf{X} - \mathbf{X}^T\mathbf{W}\mathbf{W}^T\mathbf{X} + \mathbf{X}^T\mathbf{W}\mathbf{W}^T\mathbf{W}\mathbf{W}^T\mathbf{X}) \\
&= \min_{\mathbf{W}} \text{tr}(-2\mathbf{X}^T\mathbf{W}\mathbf{W}^T\mathbf{X} + \mathbf{X}^T\mathbf{W}\mathbf{W}^T\mathbf{X}) \\
&= \min_{\mathbf{W}} \text{tr}(-\mathbf{X}^T\mathbf{W}\mathbf{W}^T\mathbf{X}) \\
&= \max_{\mathbf{W}} \text{tr}(\mathbf{X}^T\mathbf{W}\mathbf{W}^T\mathbf{X}) \tag{3.50}
\end{aligned}$$

根据矩阵迹运算的循环不变性，把式(3.50)转化为：

$$\max_{\mathbf{W}} \text{tr}(\mathbf{X}^T\mathbf{W}\mathbf{W}^T\mathbf{X}) = \max_{\mathbf{W}} \text{tr}(\mathbf{W}^T\mathbf{X}\mathbf{X}^T\mathbf{W}) \tag{3.51}$$

最终PCA的最优化问题转化为：

$$\begin{aligned}
&\max_{\mathbf{W}} \text{tr}(\mathbf{W}^T\mathbf{X}\mathbf{X}^T\mathbf{W}) \\
&\text{s. t. } \mathbf{W}^T\mathbf{W} = \mathbf{I} \tag{3.52}
\end{aligned}$$

利用拉格朗日乘子法来求解式(3.52)的最优化问题，引入新变量拉格朗日乘数 $\lambda$ ，这时只需要求解下面的拉格朗日函数的极值：

$$L(\mathbf{W}, \lambda) = \text{tr}(\mathbf{W}^T\mathbf{X}\mathbf{X}^T\mathbf{W}) + \lambda \times (\mathbf{I} - \mathbf{W}^T\mathbf{W}) \tag{3.53}$$

式(3.53)两端对 $\mathbf{W}$ 求导，并令其导数为0，可得：

$$\begin{aligned}
\frac{\partial L(\mathbf{W}, \lambda)}{\partial \mathbf{W}} &= \frac{\partial (\text{tr}(\mathbf{W}^T\mathbf{X}\mathbf{X}^T\mathbf{W}) + \lambda \times (\mathbf{I} - \mathbf{W}^T\mathbf{W}))}{\partial \mathbf{W}} = 0 \\
&\Leftrightarrow \frac{\partial (\text{tr}(\mathbf{W}^T\mathbf{X}\mathbf{X}^T\mathbf{W}))}{\partial \mathbf{W}} = \lambda \mathbf{W} \\
&\Leftrightarrow \mathbf{X}\mathbf{X}^T\mathbf{W} = \lambda \mathbf{W} \tag{3.54}
\end{aligned}$$

从式(3.54)可知， $\mathbf{W}$ 是由协方差矩阵 $\mathbf{X}\mathbf{X}^T$ 的特征向量构成的特征矩阵，由3.6节特征值与特征向量的特性可知，特征值越大，数据在其对应的特征向量的方向上所包含的信息越丰富，因此，取 $\mathbf{w}^i$ 为第 $i$ 大的特征值对应的特征向量，构成矩阵 $\mathbf{W} = \{\mathbf{w}^1, \mathbf{w}^2, \dots, \mathbf{w}^k\}$ ， $\mathbf{W} \in \mathbf{R}^{n \times k}$ ， $\mathbf{W}^T$ 就是把数据 $\mathbf{X}$ 从 $n$ 维空间映射到 $k$ 维空间的变换矩阵。PCA 的算法步骤如图3.8所示。

---

**算法3.1 PCA算法步骤**


---

输入:  $n$ 维空间的样本集合 $X = \{x^1, x^2, \dots, x^m\}$ , 其中 $x^i \in \mathbf{R}^n$ ; 映射到 $k$ 维空间

1. 预处理, 将样本的均值变为0,  $x^i = x^i - \frac{1}{m} \sum_{i=1}^m x^i$
2. 预处理, 将样本的方差变为1, 令 $\sigma = \sqrt{\frac{1}{m} \sum_{i=1}^m (x^i)^2}$ ,  $x^i = x^i / \sigma$
3. 计算协方差矩阵 $XX^T$
4. 对协方差矩阵 $XX^T$ 进行特征分解
5. 求取最大的 $k$ 个特征值, 以及特征值对应的 $k$ 个特征向量, 记为 $w^1, w^2, \dots, w^k$

输出: 投影矩阵 $W = \{w^1, w^2, \dots, w^k\}$ , 其中 $w^i \in \mathbf{R}^n$

---

图3.8 PCA的算法步骤

---

**参考文献:**


---

- [1] Kaare Brandt Petersen, Michael Syskind Pedersen. The Matrix Cookbook. 2012.
- [2] David C. Lay. Linear Algebra and Its Applications (3rd Edition)
- [3] Kenneth H. Rosen. Elementary Number Theory and its Application.
- [4] Sohrab, Houshang H. (2003). Basic Real Analysis (2nd ed.). New York: Birkhäuser. p. 104. ISBN 978-1-4939-1840-9.
- [5] Stewart, James (2008). "Chapter 15.2 Limits and Continuity". Multivariable Calculus (6th ed.). pp. 909–910. ISBN 0495011630.
- [6] C. Volinsky et al. Matrix Factorization Techniques for Recommender Systems. In IEEE Computer, Vol. 42 (2009), pp. 30-37.
- [7] Chih-Chao Ma. A Guide to Singular Value Decomposition for Collaborative Filtering. In csientuedutw (2008).

# 4

## 概率统计基础

概率论是研究随机现象和不确定性<sup>[1]</sup>的一门数学分支学科，它提供了一系列用来度量不确定性的准则和方法，以及产生新的一系列不确定性的定理。概率论是深度学习的重要基础，深度学习处理的问题都带有随机性和不确定性的特点，这主要是由于两个方面的原因造成的。

第一，数据的随机性。模型训练需要的样本数据，可以看成是从一个复杂的概率分布中随机得到，因此，学习的目的归结为尽量准确地学习到原始数据间的变量关系，并还原原始样本数据的概率分布，而从有限的训练数据中找到原始数据的分布关系本身是一个非常困难的问题。另一方面，由于从数据的获取到数据作为算法的输入使用，这中间会经历一系列的处理过程，在这个过程中，数据会受到外界环境的影响，使得数据或多或少存在各种噪声，进一步增加了学习的困难。

第二，单一模型固有的不足。任何一个学习模型都有其优缺点，在处理实际问题时，往往会训练不同模型，根据不同模型的结果帮助我们进行决策分析。不同模型对同一个测试数据可能产生不同的结果，这就需要一种衡量置信度大小的机制，帮助选择最优的结果。

本章包括概率统计和信息论相关的基础知识，它是后面概率图模型，深度学习的重要基础。如果读者对概率统计比较熟悉，可以跳过本章，如果读者想全面了解概率统计相关的知识，可以参考相关的专业教材<sup>[2]</sup>。

## 4.1 样本空间与随机变量

**样本空间：**样本空间是一个随机试验（或随机事件）所有可能结果的集合，而随机试验中的每个可能结果称为**样本点**。例如，抛一枚硬币，可能的结果有正反两面，用 $H$ 表示正面， $T$ 表示反面，即该试验的样本空间为 $\{H, T\}$ ， $H$ 和 $T$ 都是其中的样本点。

**随机变量：**设随机试验的样本空间为集合 $S = \{e\}$ ，随机变量 $X$ 本质上是一个实值函数，它为每一个试验的结果都分配一个实数值。仍以抛硬币为例，如果把正面分配为1，反面分配为0，则随机变量 $x$ 的定义可以写成 $X(H) = 1$ ， $X(T) = 0$ 。本书通常采用大写拉丁字母来表示随机变量，如 $X$ 、 $Y$ 等。

根据随机变量的取值范围不同，可以把随机变量区分为离散随机变量和连续随机变量（其他类型的随机变量本书不做讨论）。

**离散随机变量：**若随机变量的取值范围是有限个或者可列无限个，这种随机变量称为离散随机变量。要注意的是离散随机变量的取值不一定是整数。

**连续随机变量：**对于随机变量 $X$ 的分布函数为 $F(x)$ ，若存在非负函数 $f(x)$ ，使得对任意的实数 $x$ ，满足：

$$F(X) = P(X \leq x) = \int_{-\infty}^x f(t)dt \quad (4.1)$$

则称随机变量 $X$ 为连续型随机变量，其中 $f(x)$ 被称为 $X$ 的概率密度函数，有关分布函数和概率密度函数的定义将在后两节讲解。

## 4.2 概率分布与分布函数

概率分布是描述一维随机变量或者多维随机变量在任意一个取值上的可能性大小，对于一维随机变量 $X$ ，其概率分布通常记为 $P(X=x)$ ，或 $X \sim P(x)$ （表示 $X$ 服从概率分布 $P(x)$ ）。

概率分布描述了单一取值的可能性，但在实际应用中，很多时候我们并不关心取某一个值的概率，特别是对于连续型随机变量，我们后面会看到它在某一点的概率都为0，因此，我们通常比较关心的是随机变量落在某一区间的概率，为此，引入分布函数的概念。

定义：设 $X$ 是一个随机变量， $x_k$ 是任意实数值，函数：

$$F(x_k) = P(X \leq x_k) \quad (4.2)$$

称为随机变量 $X$ 的**分布函数**。

观察式(4.2)可以发现，对于任意的实数 $x_1, x_2 (x_1 < x_2)$ ，有：

$$\begin{aligned} P(x_1 < X \leq x_2) &= P(X \leq x_2) - P(X \leq x_1) \\ &= F(x_2) - F(x_1) \end{aligned} \quad (4.3)$$

成立。式(4.3)表明，若随机变量 $X$ 的分布函数已知，那么可以知道 $X$ 落入任意一个区间 $[x_1, x_2]$ 的概率。

## 4.3 一维随机变量

对于随机变量，根据其维度大小的不同可以划分为一维随机变量和多维随机变量。本节将探讨最简单的一维随机变量的统计性质，一维随机变量按照取值的范围不同又可以分为离散型随机变量和连续型随机变量。

### 4.3.1 离散型随机变量和分布律

离散型随机变量采用分布律来描述变量的概率分布。

设离散型随机变量 $X$ 的所有可能取值为 $x_k (k = 1, 2, \dots)$ ， $X$ 取各个可能值的概率，即事件 $\{X = x_k\}$ 的概率为：

$$P(X = x_k) = p_k, \quad k = 1, 2, \dots \quad (4.4)$$

其中， $p_k$ 满足下面的两个条件。

(1)  $0 \leq p_k \leq 1, \quad k = 1, 2, \dots$ 。

(2)  $\sum_{k=1}^{\infty} p_k = 1$ 。

我们称式(4.4)为离散型随机变量 $X$ 的**分布律**，也可以用表格来表示分布律，如表4.1所示。

表4.1 离散随机变量分布律

$X$	$x_1$	$x_2$	$\cdots$	$x_i$	$\cdots$
$P(X = x_k)$	$p_1$	$p_2$	$\cdots$	$p_i$	$\cdots$

由概率的可列可加性可得随机变量 $X$ 的分布函数为：

$$F(x) = P(X \leq x) = \sum_{x_k \leq x} P(X = x_k) \quad (4.5)$$

从式(4.5)可以看到，离散型随机变量的分布函数 $F(x)$ 是一个阶梯型函数，其图像如图4.1所示。

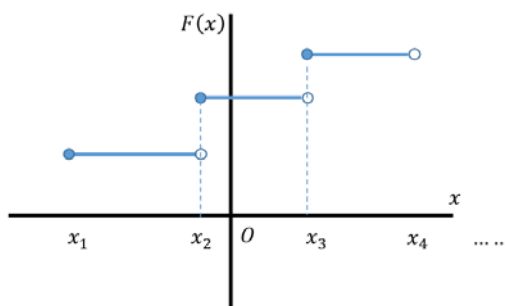


图 4.1 一维离散型随机变量分布函数图像

### 4.3.2 连续型随机变量和概率密度函数

与离散型随机变量不同，连续型随机变量采用概率密度函数来描述变量的概率分布。

设有随机变量 $X$ ，对于函数 $f(x)$ ，当它满足下面三个性质时，我们称 $f(x)$ 为概率密度函数。

- (1)  $f(x) \geq 0$ ，注意这里不要求 $f(x) \leq 1$ 。
- (2)  $\int_{-\infty}^{\infty} f(x) dx = 1$ 。
- (3) 对于任意的实数 $x_1$ 和 $x_2$ ，且 $x_1 \leq x_2$ ，有：

$$P(x_1 < X \leq x_2) = \int_{x_1}^{x_2} f(x) dx \quad (4.6)$$

成立。

性质(2)表明，概率密度函数 $y = f(x)$ 与 $x$ 轴之间的面积等于1，如图4.2(a)所示。而性质(3)表明 $X$ 落在区间 $[x_1, x_2]$ 的概率 $P(x_1 \leq X \leq x_2)$ 等于区间 $[x_1, x_2]$ 与曲线 $y = f(x)$ 构成的曲边梯形的面积，如图4.2(b)所示。

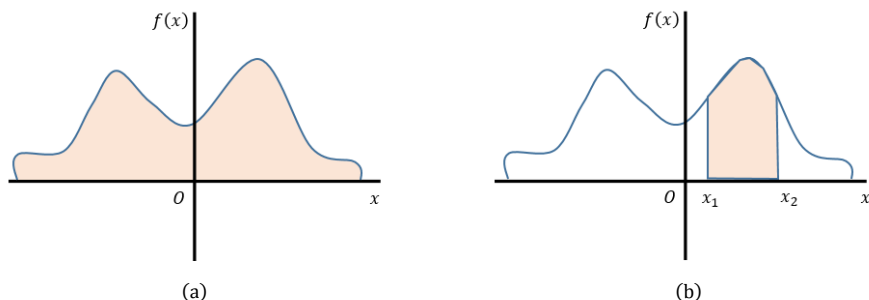


图4.2 概率密度函数

验证4.2节提到的一个重要结论：连续型随机变量在任意一点处的概率处处为0。

设有一任意小的实数 $\Delta x$ ，由于 $\{X = x\} \subset \{x - \Delta x < X \leq x\}$ ，根据式(4.2)分布函数的定义，可得：

$$0 \leq P(X = x) \leq P(x - \Delta x < X \leq x) = F(x) - F(x - \Delta x) \quad (4.7)$$

令 $\Delta x \rightarrow 0$ ，由夹逼准则，式(4.7)可求得：

$$P(X = x) = 0 \quad (4.8)$$

式(4.8)表明，对于连续型随机变量，它在任意一点的取值的概率都为0。

因此，在连续型随机变量中，当讨论区间的概率定义时，一般对开区间和闭区间不加以区分，即等式 $P(x_1 \leq X \leq x_2) = P(x_1 < X \leq x_2) = P(x_1 \leq X < x_2) = P(x_1 < X < x_2)$ 成立。

## 4.4 多维随机变量

设有 $n$ 个随机试验 $S_1, S_2, \dots, S_n$ ，它们的样本空间分别为： $S_1 = \{e_1\}, S_2 = \{e_2\}, \dots, S_n = \{e_n\}$ ，设 $X^1(e_1), X^2(e_2), \dots, X^n(e_n)$ 是分别定义在随机试验 $S_1, S_2, \dots, S_n$ 上的随机变量，由它们构成的一个向量 $(X^1, X^2, \dots, X^n)$ 叫作 $n$ 维随机变量。

为了讨论方便，后面讨论多维随机变量时，除非特别说明，否则都是以二维随机



变量为例,记为 $(X,Y)$ ,但所讨论的内容同样适用于多维随机变量。

二维随机变量的性质不仅与 $X$ 和 $Y$ 有关,还与 $X$ 和 $Y$ 的相互关系有关,因此,不但要研究 $X$ 或者 $Y$ 的各自性质,还需要将 $X$ 和 $Y$ 当成一个整体看待,本节先来讨论整体的联合分布,后面两节将分别讨论边缘分布和条件分布。

#### 4.4.1 离散型二维随机变量和联合分布律

4.3节介绍了一维离散型随机变量的概率分布,对于二维离散型随机变量,我们采用联合分布律来描述二维向量的概率分布。

设二维离散随机变量 $(X,Y)$ 的所有可能取值为 $(x_i, y_j)(i, j = 1, 2, \dots)$ ,把

$$P(X = x_i, Y = y_j) = p_{ij}, \quad i, j = 1, 2, \dots \quad (4.9)$$

称为联合分布律,其中, $p_{ij}$ 满足下面的两个条件。

(1)  $0 \leq p_{ij} \leq 1, \quad i, j = 1, 2, \dots$ 。

(2)  $\sum_{i=1}^{\infty} \sum_{j=1}^{\infty} p_{ij} = 1$ 。

有时也可以用表格来表示二维离散随机变量 $(X,Y)$ 的联合分布律,如表4.2所示。

表 4.2 二维随机变量分布律

$\begin{matrix} x \\ y \end{matrix}$	$x_1$	$x_2$	$\dots$	$x_i$	$\dots$
$y_1$	$p_{11}$	$p_{21}$	$\dots$	$p_{i1}$	$\dots$
$y_2$	$p_{12}$	$p_{22}$	$\dots$	$p_{i2}$	$\dots$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	
$y_j$	$p_{1j}$	$p_{2j}$	$\dots$	$p_{ij}$	$\dots$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	

#### 4.4.2 连续型二维随机变量和联合密度函数

与一维连续随机变量类似,对于二维随机变量 $(X,Y)$ 的分布函数 $F(x,y)$ ,如果存在非负的函数 $f(x,y)$ ,使得对任意的 $x,y$ 有:

$$F(x, y) = \int_{-\infty}^y \int_{-\infty}^x f(u, v) du dv \quad (4.10)$$

成立, 则  $f(x, y)$  称为二维随机变量  $(X, Y)$  的联合密度函数, 联合密度函数  $f(x, y)$  同样具有下面的几个性质。

(1)  $f(x, y) \geq 0$ , 注意这里不要求  $f(x, y) \leq 1$ 。

(2)  $\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) dx dy = 1$ 。

(3) 设  $D$  是  $OXY$  平面的任意一个区域, 点  $(x, y)$  落在该区域的概率为:

$$P((x, y) \in D) = \iint_D f(x, y) dx dy \quad (4.11)$$

从几何的角度来看, 概率密度函数  $z = f(x, y)$  表示空间的一个曲面。性质(2)表明, 介于曲面  $z = f(x, y)$  与  $OXY$  平面之间的空间区域的体积为1。性质(3)表明点  $(x, y)$  落在平面区域  $D$  的体积为: 以区域  $D$  为底,  $z = f(x, y)$  为顶面的柱体的体积, 如图4.3所示。

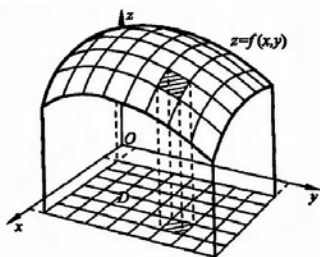


图4.3 联合密度函数 (图片摘自华东师范大学《数学分析》, 第三版)

## 4.5 边缘分布

对二维随机变量  $(X, Y)$ , 现有随机变量  $X$  与随机变量  $Y$ , 其联合分布函数为  $F(x, y)$ , 而  $X$  与  $Y$  各自的分布函数分别为  $F_X(x)$  和  $F_Y(y)$ , 它们分别称为二维随机变量  $(X, Y)$  关于  $X$  和关于  $Y$  的边缘分布函数。

对于离散型二维随机变量  $(X, Y)$ , 它的边缘分布律分别定义为:

$$P(X = x_i) = \sum_{y_j} P(X = x_i, Y = y_j) = p_{i.} \quad (4.12)$$

$$P(Y = y_j) = \sum_{x_i} P(X = x_i, Y = y_j) = p_{.j} \quad (4.13)$$

对于连续型二维随机变量 $(X, Y)$ ，它的边缘密度函数分别为：

$$f(x) = \int_{-\infty}^{\infty} f(x, y) dy \quad (4.14)$$

$$f(y) = \int_{-\infty}^{\infty} f(x, y) dx \quad (4.15)$$

同理，由分布函数的定义，可以得到离散型随机变量与连续型随机变量的边缘分布函数。

设二维随机变量 $(X, Y)$ ，对于离散型随机变量：

$$F_X(x) = F(x, \infty) = \sum_{x_i \leq x} \sum_{j=1}^{\infty} p_{ij} \quad (4.16)$$

$$F_Y(y) = F(\infty, y) = \sum_{y_j \leq y} \sum_{i=1}^{\infty} p_{ij} \quad (4.17)$$

对于连续型随机变量：

$$F_X(x) = F(x, \infty) = \int_{-\infty}^x \left[ \int_{-\infty}^{\infty} f(x, y) dy \right] dx \quad (4.18)$$

$$F_Y(y) = F(\infty, y) = \int_{-\infty}^y \left[ \int_{-\infty}^{\infty} f(x, y) dx \right] dy \quad (4.19)$$

## 4.6 条件分布与链式法则

条件分布和链式法则是概率论中非常重要的公式，也是很多机器学习算法模型的理论基础，如朴素贝叶斯模型就分别用到了条件分布和链式法则来推导。

### 4.6.1 条件概率

设有两个随机变量 $X, Y$ ，且 $P(X) > 0$ ，称：

$$P(Y|X) = \frac{P(X \times Y)}{P(X)} \quad (4.20)$$

为在事件 $X$ 发生的条件下事件 $Y$ 发生的条件概率。

由条件概率我们很容易得到如下**条件分布函数**的概念。

对于二维离散随机变量 $(X, Y)$ ：

对于固定的 $j$ ，若 $P(Y = y_j) > 0$ ，则称

$$P(X = x_i | Y = y_j) = \frac{P(X = x_i, Y = y_j)}{P(Y = y_j)} = \frac{p_{ij}}{p_{.j}} \quad (4.21)$$

为在 $Y = y_j$ 的条件下随机变量 $X = x_i$ 的**条件分布律**。

对于二维连续随机变量 $(X, Y)$ ：

连续随机变量在任意一点的取值 $P(X = x) = 0$ ，因此不能像离散型随机变量那样直接导出连续随机变量的条件密度函数。

与4.4.2节类似，通过先引入条件分布函数来得到条件密度函数，设二维连续随机变量 $(X, Y)$ 的密度函数为 $f(x, y)$ ，关于 $Y$ 的边缘概率密度为 $f_Y(y)$ ，给定 $y$ ，对于任意固定的 $\epsilon > 0$ ，考虑下面的概率：

$$\begin{aligned} P(X \leq x | y < Y \leq y + \epsilon) &= \frac{P(X \leq x, y < Y \leq y + \epsilon)}{P(y < Y \leq y + \epsilon)} \\ &= \frac{\int_{-\infty}^x \left[ \int_y^{y+\epsilon} f(x, y) dy \right] dx}{\int_y^{y+\epsilon} f_Y(y) dy} \end{aligned} \quad (4.22)$$

在连续的条件下，当 $\epsilon \rightarrow 0$ ，式(4.22)的分子、分母可以分别近似为：

$$\lim_{\epsilon \rightarrow 0} \int_{-\infty}^x \left[ \int_y^{y+\epsilon} f(x, y) dy \right] dx = \epsilon \times \int_{-\infty}^x f(x, y) dx \quad (4.23)$$

$$\lim_{\epsilon \rightarrow 0} \int_y^{y+\epsilon} f_Y(y) dy = \epsilon \times f_Y(y) \quad (4.24)$$

把式(4.23)与式(4.24)代入式(4.22)可得：

$$P(X \leq x | y < Y \leq y + \epsilon) = \frac{\epsilon \times \int_{-\infty}^x f(x, y) dx}{\epsilon \times f_Y(y)} = \int_{-\infty}^x \frac{f(x, y)}{f_Y(y)} dx \quad (4.25)$$

由上式，给出条件密度函数和条件分布函数的定义：

设二维随机变量 $(X, Y)$ 的密度函数为 $f(x, y)$ ，关于 $Y$ 的边缘概率密度为 $f_Y(y)$ ，且满

足  $f_Y(y) > 0$ , 记:

$$f_{X|Y}(x|y) = \frac{f(x, y)}{f_Y(y)} \quad (4.26)$$

为在  $Y = y$  的条件下, 关于  $X$  的条件密度函数, 把

$$F_{X|Y}(x|y) = \int_{-\infty}^x f_{X|Y}(x|y) dx \quad (4.27)$$

称为在  $Y = y$  的条件下, 关于  $X$  的条件分布函数。

同理, 可以得到在  $X = x$  的条件下, 关于  $Y$  的条件密度函数为:

$$f_{Y|X}(y|x) = \frac{f(x, y)}{f_X(x)} \quad (4.28)$$

在  $X = x$  的条件下, 关于  $Y$  的条件分布函数为:

$$F_{Y|X}(y|x) = \int_{-\infty}^y f_{Y|X}(y|x) dy \quad (4.29)$$

## 4.6.2 链式法则

条件概率的链式法则, 也称为乘法法则, 把式(4.20)变形可以得到条件概率的乘法法则:

$$P(AB) = P(A) \times P(B|A) \quad (4.30)$$

把上式推广到  $n$  维随机变量, 可得:

$$P(X^1, X^2, \dots, X^n) = P(X^1) \times \prod_{i=2}^n P(X^i | X^1, \dots, X^{i-1}) \quad (4.31)$$

## 4.7 多维随机变量的独立性分析

在求解机器学习算法时, 很多时候都会遇到时间复杂度很高而难以训练的情况, 这时候一方面可以对算法进行改进, 另一方面, 也可以在不影响结果的前提下做一些简化计算的假设。这其中独立性假设是很多机器学习模型能够高效训练的基础性假设, 在下一章的学习中, 读者将会发现, 独立性假设贯穿了整个概率图模型的理论分析。

### 4.7.1 边缘独立

定义：设 $F(x, y)$ 、 $F_X(x)$ 和 $F_Y(y)$ 分别是三维随机变量 $(X, Y)$ 的联合分布函数及边缘分布函数，如果对于所有的 $x$ 、 $y$ ，有：

$$F(x, y) = F_X(x) \times F_Y(y) \quad (4.32)$$

成立，则称随机变量 $X$ 与 $Y$ 相互独立，或边缘独立，记作 $X \perp Y$ 。

对于离散型随机变量，式(4.32)等价于对二维随机变量 $(X, Y)$ 的所有可能取值 $(x_i, y_j)$ ，我们有：

$$P(X = x_i, Y = y_j) = P(X = x_i) \times P(Y = y_j) \quad (4.33)$$

成立。

对于连续型随机变量，设 $f(x, y)$ 、 $f_X(x)$ 和 $f_Y(y)$ 分别是二维随机变量 $(X, Y)$ 的联合概率密度函数及边缘密度函数，则式(4.32)等价于：

$$f(x, y) = f_X(x) \times f_Y(y) \quad (4.34)$$

成立。

在实际应用中，式(4.33)与式(4.34)要比式(4.32)更为方便。

### 4.7.2 条件独立

定义：现有三维随机变量 $(X, Y, Z)$ ，设：

$F_{Y \times Z|X}(y \times z|x)$ ：在 $X = x$ 的条件下， $Y \times Z$ 的联合分布函数。

$F_{Y|X}(y|x)$ ：在 $X = x$ 的条件下， $Y$ 的条件分布函数。

$F_{Z|X}(z|x)$ ：在 $X = x$ 的条件下， $Z$ 的条件分布函数。

如果对于所有的 $x$ 、 $y$ 、 $z$ ，有：

$$F_{Y \times Z|X}(y \times z|x) = F_{Y|X}(y|x) \times F_{Z|X}(z|x) \quad (4.35)$$

成立，则称在给定条件 $X$ 下，随机变量 $Y$ 与随机变量 $Z$ 相互独立，记作 $(Y \perp Z|X)$ 。

对于离散型随机变量，式(4.35)等价于：对二维随机变量 $(X, Y, Z)$ 的所有可能取值 $(x_i, y_j, z_k)$ ，有：

$$P(Y = y_j, Z = z_k | X = x_i) = P(Y = y_j | X = x_i) \times P(Z = z_k | X = x_i) \quad (4.36)$$

成立。

对于连续型随机变量，设：

$f_{Y*Z|X}(y \times z | x)$ ：在  $X = x$  的条件下， $Y \times Z$  的联合概率密度函数。

$f_{Y|X}(y | x)$ ：在  $X = x$  的条件下， $Y$  的概率密度函数。

$f_{Z|X}(z | x)$ ：在  $X = x$  的条件下， $Z$  的概率密度函数。

则式(4.35)等价于：

$$f_{Y*Z|X}(y \times z | x) = f_{Y|X}(y | x) \times f_{Z|X}(z | x) \quad (4.37)$$

成立。

## 4.8 数学期望、方差、协方差

概率统计的一个很重要的应用是如何对数据进行分析，并找出数据间潜在的规律。本节讨论统计学中常用的三个统计量，即数学期望、方差和协方差。

### 4.8.1 数学期望

设随机变量  $X$  服从分布  $P(X)$ ，数学期望是指随机试验中，每次可能结果的概率  $P(x)$  乘以其结果  $x$  的总和。数学期望是反映随机变量平均取值的大小的统计量。

对于离散型随机变量  $X$ ，设其分布律为：

$$P(X = x_k) = p_k, \quad k = 1, 2, \dots$$

若级数  $\sum_{k=1}^{\infty} x_k \times p_k$  绝对收敛，则称级数  $\sum_{k=1}^{\infty} x_k \times p_k$  的值为随机变量  $X$  的数学期望，记为：

$$E(X) = \sum_{k=1}^{\infty} x_k \times p_k \quad (4.38)$$

对于连续型随机变量  $X$ ，设其概率密度函数为  $f(x)$ ，若积分：

$$\int_{-\infty}^{\infty} x f(x) dx \quad (4.39)$$

绝对收敛，则把该积分的值称为随机变量 $X$ 的数学期望，记为：

$$E(X) = \int_{-\infty}^{\infty} xf(x)dx \quad (4.40)$$

### 4.8.2 方差

定义：现有随机变量 $X$ ，若 $E([X - E(X)]^2)$ 存在，则称 $E([X - E(X)]^2)$ 为随机变量的方差，记为

$$\text{Var}(x) = E([X - E(X)]^2) \quad (4.41)$$

方差是度量随机变量与其数学期望之间的偏离程度或分散程度的统计量。方差越小，则数据越集中，相反，方差越大，则数据越分散。

### 4.8.3 协方差

协方差是衡量多维随机变量之间的相关性的一种统计量，对随机变量 $X$ 与随机变量 $Y$ ，定义其协方差为：

$$\text{Cov}(X, Y) = E([X - E(X)][Y - E(Y)]) \quad (4.42)$$

化简上式，可得协方差的另一种表示方法：

$$\text{Cov}(X, Y) = E(XY) - E(X) \times E(Y) \quad (4.43)$$

方差只是衡量一个变量与期望之间的偏离程度，而协方差则是衡量两个变量的线性相关性，当 $X = Y$ 时，协方差的结果就等于方差。

当协方差大于0时，则表示随机变量 $X$ 与随机变量 $Y$ 是正相关，也就是说，它们具有相同的变化趋势，如图4.4所示。



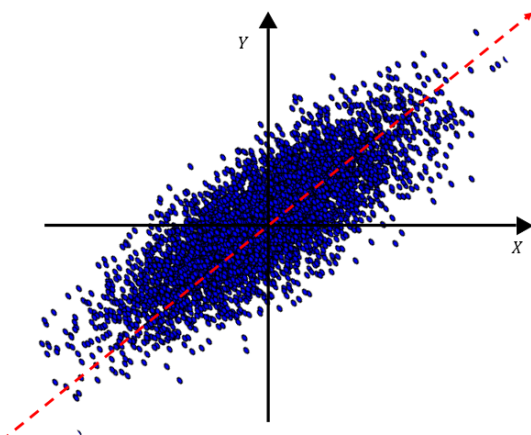


图4.4 正相关

当协方差小于0时,则表示随机变量 $X$ 与随机变量 $Y$ 是负相关,也就是说,它们具有相反的变化趋势,如图4.5所示。

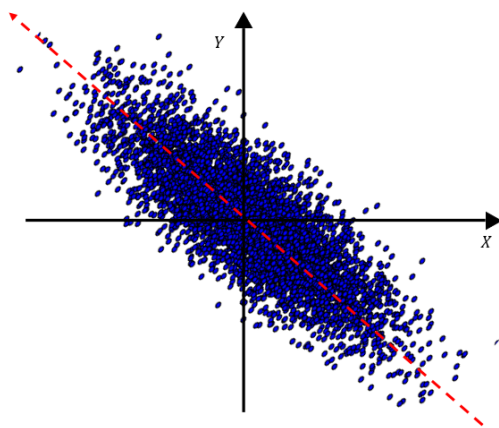


图4.5 负相关

当协方差等于0时,则表示随机变量 $X$ 与随机变量 $Y$ 没有线性相关性,如图4.6所示。

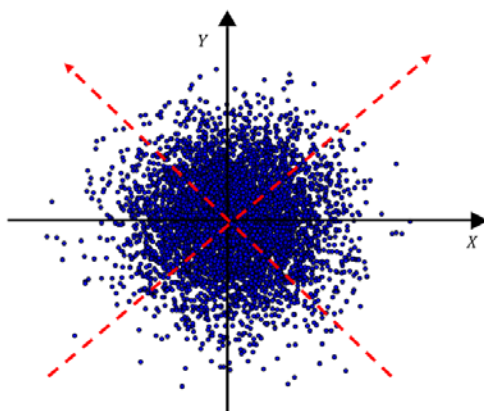


图4.6 线性无关

我们应该区分**线性不相关**与**独立**的区别，如果随机变量 $X$ 与随机变量 $Y$ 相互独立，那么两个变量的协方差必为0，即两者是线性不相关的；但反之不成立，当两个随机变量是线性不相关时，随机变量之间不一定相互独立，因为独立性考察的是一般关系，而协方差只是度量线性关系。

协方差描述了两个随机变量间的正负线性相关性，而相关系数通过归一化，提供了一种衡量相关性大小的统计量，相关系数的计算公式如下：

$$\rho_{XY} = \frac{\text{Cov}(X, Y)}{\sqrt{\text{Var}(X)} \times \sqrt{\text{Var}(Y)}} \quad (4.44)$$

从式(4.44)可以看到，相关系数是在协方差的基础上添加了正则化因子，从而把相关系数的值限定与 $[-1, 1]$ 区间上，当 $\rho_{XY} = 1$ 时，随机变量 $X$ 与随机变量 $Y$ 是线性相关的，即可以表示为 $Y = a \times X + b$ ，其中 $a$ 和 $b$ 是任意实数，且 $a > 0$ ；当 $\rho_{XY} = -1$ ，则随机变量 $X$ 与随机变量 $Y$ 是负线性相关的，即可以表示为 $Y = -a \times X + b$ ，其中 $a$ 和 $b$ 是任意实数，且 $a > 0$ 。

#### 4.8.4 协方差矩阵

在统计学中，为了进一步探索多维变量之间的关系，在协方差的基础之上，提出了协方差矩阵的概念，设有 $n$ 个随机变量 $(X_1, X_2, \dots, X_n)$ ，定义其协方差矩阵为：

$$\mathbf{C} = \begin{pmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{n1} & \cdots & c_{nn} \end{pmatrix} \quad (4.45)$$

其中 $c_{ij}$ 表示随机变量 $X_i$ 与随机变量 $X_j$ 的协方差,由协方差的定义可以得到矩阵 $\mathbf{C}$ 的重要性质。

第一,若协方差矩阵是一个对称矩阵,则 $c_{ij} = c_{ji}$ 。

第二,矩阵 $\mathbf{C}$ 的对角线元素都大于或等于0,值等于每一个随机变量的方差,即 $c_{ii} = \text{Var}(X_i)$ 。

设有随机变量 $X_1$ 和随机变量 $X_2$ ,下面来分析不同的数据分布与协方差矩阵之间的对应关系。

- 当协方差矩阵如下所示:

$$\mathbf{C} = \begin{pmatrix} 5 & 4 \\ 4 & 6 \end{pmatrix}$$

观察协方差矩阵,  $c_{12} = 4$ ,说明随机变量 $X_1$ 与随机变量 $X_2$ 是正相关,且随机变量 $X_1$ 的方差为5,随机变量 $X_2$ 的方差为6,说明随机变量 $X_1$ 与随机变量 $X_2$ 各自的数据也比较分散,如图4.7所示。

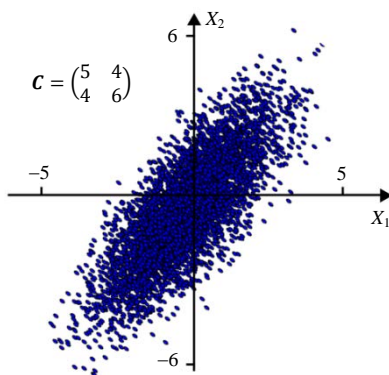


图4.7

- 当协方差矩阵如下所示:

$$\mathbf{C} = \begin{pmatrix} 5 & -4 \\ -4 & 6 \end{pmatrix}$$

观察协方差矩阵,  $c_{12} = -4$ ,说明随机变量 $X_1$ 与随机变量 $X_2$ 是负相关,且随机变量 $X_1$ 的方差为5,随机变量 $X_2$ 的方差为6,说明随机变量 $X_1$ 与随机变量 $X_2$ 各自的数据也比较分散,如图4.8所示。

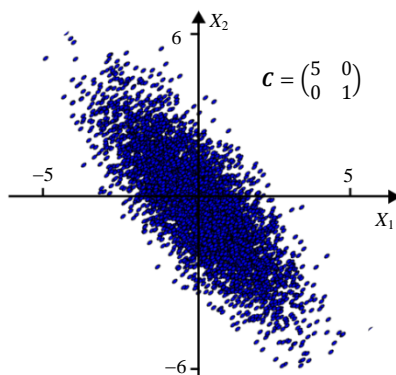


图4.8

- 当协方差矩阵如下所示：

$$c = \begin{pmatrix} 5 & 0 \\ 0 & 1 \end{pmatrix}$$

观察协方差矩阵， $c_{12} = 0$ ，说明随机变量 $X_1$ 与随机变量 $X_2$ 没有线性相关性，且随机变量 $X_1$ 的方差为5，随机变量 $X_2$ 的方差为1，说明数据在 $X_1$ 的方向上会比较发散，而在 $X_2$ 的方向上会比较集中，如图4.9所示。

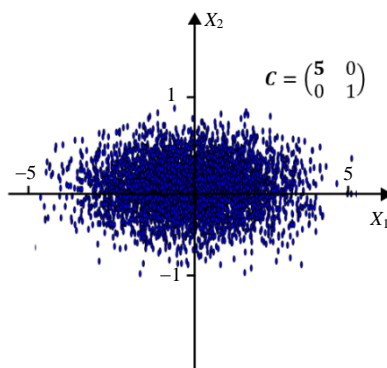


图4.9

- 当协方差矩阵如下所示：

$$c = \begin{pmatrix} 1 & 0 \\ 0 & 5 \end{pmatrix}$$

观察协方差矩阵， $c_{12} = 0$ ，说明随机变量 $X_1$ 与随机变量 $X_2$ 没有线性相关性，且随机变量 $X_1$ 的方差为1，随机变量 $X_2$ 的方差为5，说明数据在 $X_1$ 的方向上会比较集中，而在 $X_2$ 的方向上会比较发散，如图4.10所示。

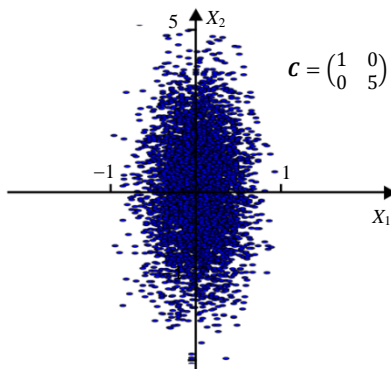


图4.10

## 4.9 信息论基础

本节将转向信息论相关的基础知识，它是应用数学的一个分支学科，它把信息的传递作为一种随机现象来考虑，给出了估算通信信道容量的方法。

本节将重点介绍5个相关概念，分别是信息熵、条件熵、互信息、交叉熵和相对熵。它们在深度学习和人工智能的其他领域都被广泛采用，因此，我们也会分别给出它们在深度学习领域常用的使用场景。

此外，需要强调一点的是，本书中把信息作为随机变量来处理，一般指的是**离散型随机变量**。

### 4.9.1 信息熵

在信息论和统计学中，信息熵简称为熵，是表示随机变量不确定性的度量，在1948年，克劳德·艾尔伍德·香农将热力学的熵引入到信息论，因此它又被称为香农熵。

设 $X$ 是一离散型随机变量，其概率分布为：

$$P(X = x_k) = p_k, \quad k = 1, 2, \dots, n \quad (4.46)$$

则随机变量的信息熵定义为：

$$H(X) = - \sum_{i=1}^n p_i \times \log p_i \quad (4.47)$$

$\log p_i$ 表示以2为底的对数。先规定一些边界取值，当 $p_i = 0$ 时，定义 $0 \times \log 0 = 0$ ，

这个取值在本书后面同样适用。

信息熵越大，包含的信息就越多，那么随机变量的不确定性就越大。例如，假设随机变量 $X$ 服从0-1分布，即概率分布为：

$$P(X=1)=p, \quad P(X=0)=1-p, \quad 0 \leq p \leq 1$$

这时，熵的表达式为：

$$\begin{aligned} H(X) &= - \sum_{i=1}^n p_i \times \log p_i \\ &= -p \times \log p - (1-p) \times \log p(1-p) \end{aligned} \quad (4.48)$$

熵 $H(X)$ 随着概率 $p$ 变化的曲线如图4.11所示。

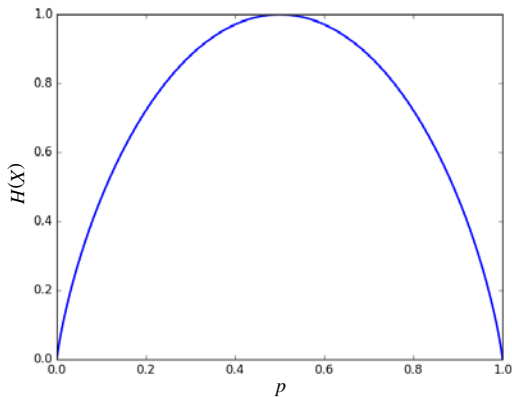


图4.11 当随机变量 $X$ 服从0-1分布时，熵与概率的关系图

从图中可以看到，当 $p=0$ 或 $p=1$ 时，随机变量没有任何的不确定性，因此它的熵位于最小值；当 $p=0.5$ 时，熵的取值最大，随机变量的不确定性也最大。下面给出信息熵的极值最大熵定理。

**最大熵定理：**当离散随机变量的概率分布是等概率分布时， $H(X)$ 取最大值，结果为 $\log_2 n$ ，其中 $n$ 表示随机变量 $X$ 有 $n$ 个不同的取值。

证：当离散随机变量 $X$ 的概率分布是等概率分布时， $p_i = 1/n, i = 1, 2, \dots, n$ 。即有：

$$H(X) = - \sum_{i=1}^n \frac{1}{n} \times \log \frac{1}{n} = \log_2 n$$

### 4.9.2 条件熵

设有二维随机变量 $(X, Y)$ ，其联合的概率密度为：

$$P(X = x_i, Y = y_j) = p_{ij}, \quad i = 1, 2, \dots, n; \quad j = 1, 2, \dots, m \quad (4.49)$$

条件熵 $H(Y|X)$ 表示在已知随机变量 $X$ 的条件下，随机变量 $Y$ 的不确定性，它的计算公式为：

$$H(Y|X) = - \sum_{i=1}^n \sum_{j=1}^m p(X = x_i, Y = y_j) \times \log p(Y = y_j | X = x_i) \quad (4.50)$$

从感知上来说，条件熵的值要比信息熵小，因为当我们有了更多的背景知识时，信息的不确定性自然也就下降了。例如，在语言模型中，二元模型要比一元模型好，当我们知道前面的一个单词时，当前位置的单词可取的范围自然小很多，也就是不确定性会降低。

下面给出形式化的定理来证明上面的结论。

定理：对二维随机变量 $(X, Y)$ ，条件熵 $H(Y|X)$ 和信息熵 $H(Y)$ 的关系满足

$$H(Y|X) \leq H(Y) \quad (4.51)$$

证：

$$\begin{aligned} H(Y|X) &= - \sum_{i=1}^n \sum_{j=1}^m p(X = x_i, Y = y_j) \times \log p(Y = y_j | X = x_i) \\ &= - \sum_{i=1}^n p(X = x_i) \left[ \sum_{j=1}^m p(Y = y_j | X = x_i) \times \log p(Y = y_j | X = x_i) \right] \\ &\leq - \sum_{i=1}^n p(X = x_i) \left[ \sum_{j=1}^m p(Y = y_j | X = x_i) \times \log p(Y = y_j) \right] \\ &= - \sum_{j=1}^m \left[ \sum_{i=1}^n p(Y = y_j | X = x_i) * p(X = x_i) \right] \times \log p(Y = y_j) \\ &= - \sum_{j=1}^m p(Y = y_j) \times \log p(Y = y_j) \\ &= H(Y) \end{aligned}$$

当随机变量 $X$ 与随机变量 $Y$ 相互独立时， $H(Y|X) = H(Y)$ 。从感知上来说，随机变

量 $X$ 的信息对理解随机变量 $Y$ 没有帮助，没有消除不确定性。

### 4.9.3 互信息

互信息，也成为信息增益，是描述两个随机变量之间的相关性程度，也就是给定一个随机变量 $X$ 后，另一个随机变量 $Y$ 不确定性的削弱程度，记为：

$$I(X, Y) = H(Y) - H(Y|X) \quad (4.52)$$

由4.9.1节和4.9.2节的知识，我们可以得到互信息的几个相关性质。

- 因为 $H(Y) \geq H(Y|X)$ ，所以 $I(X, Y)$ 的取值范围为 $0 \leq I(X, Y) \leq H(Y)$ 。
- 当随机变量 $X$ 与随机变量 $Y$ 完全相关时，条件熵 $H(Y|X) = 0$ ，这时 $I(X, Y)$ 取最大值。
- 当随机变量 $X$ 与随机变量 $Y$ 完全无关时，条件熵 $H(Y|X) = H(Y)$ ，这时 $I(X, Y)$ 取最小值。

在决策树算法中，互信息被用来作为特征选取的一种度量指标，给定训练数据集 $D$ ，每个数据集都由 $n$ 维特征构成，构建决策树时，一个核心的问题是采用哪一个特征来划分数数据集？每一个特征可以看成是一个随机变量， $n$ 维特征可以记为 $(X_1, X_2, \dots, X_n)$ 。

一种合理的选择方案是，分别计算 $I(D, X_i)$ ，计算第 $i$ 维特征与训练数据集 $D$ 的相关性， $I(D, X_i)$ 越大，说明第 $i$ 维特征与训练数据集 $D$ 越相关，也就是第 $i$ 维特征的数据包含数据集 $D$ 的信息更多。

### 4.9.4 相对熵与交叉熵

在本章的开头我曾经提到，机器学习和深度学习的目的归结为尽量准确的学习到数据间的变量关系，还原样本数据的概率分布。交叉熵和相对熵正是衡量概率分布或者函数之间相似性的度量方法。

设有随机变量 $X$ ，其真实概率分布为 $p(x)$ ，通过模型训练得到的概率分布为 $q(x)$ ，下面分析如何通过交叉熵和相对熵来衡量 $q(x)$ 与 $p(x)$ 的相似性。

#### 1. 相对熵

相对熵，全称为*Kullback-Leibler Divergence*，也被称为KL散度，KL距离，其定义为：



$$\text{KL}(p(x)||q(x)) = \sum_{x \in X} p(x) \times \log_2 \left( \frac{p(x)}{q(x)} \right) \quad (4.53)$$

下面我们来看看相对熵具有的一些重要性质。

(1) 相对熵不是传统意义上的“距离”，这是因为相对熵不具有对称性，即  $\text{KL}(p(x)||q(x)) \neq \text{KL}(q(x)||p(x))$ 。

(2) 当预测的分布  $q(x)$  与真实概率分布  $p(x)$  完全相同时，相对熵  $\text{KL}(p(x)||q(x)) = 0$ 。

(3) 如果两个分布的差异越大，那么相对熵越大；反之，两个分布的差异越小，那么相对熵越小。

(4) 相对熵满足非负性，即  $\text{KL}(p(x)||q(x)) \geq 0$ 。

首先简要证明性质(4)。

我们知道，对于  $x > 0$  有不等式  $\log_2 x \leq x - 1$  成立，如图4.12所示。

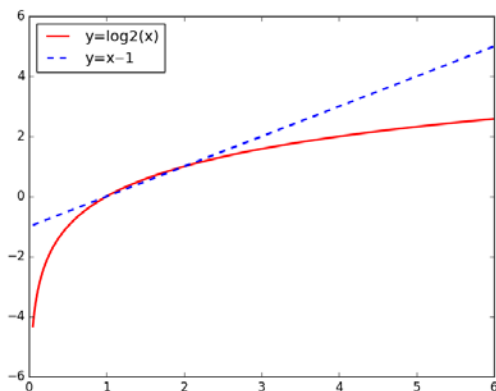


图4.12

故：

$$\begin{aligned} \text{KL}(p(x)||q(x)) &= \sum_{x \in X} p(x) \times \log_2 \left( \frac{p(x)}{q(x)} \right) = - \sum_{x \in X} p(x) \times \log_2 \left( \frac{q(x)}{p(x)} \right) \\ &\geq - \sum_{x \in X} p(x) \times \left( \left( \frac{q(x)}{p(x)} \right) - 1 \right) = - \sum_{x \in X} p(x) \times \left( \left( \frac{q(x)}{p(x)} \right) - 1 \right) \\ &= - \left[ \sum_{x \in X} p(x) - \sum_{x \in X} q(x) \right] = 0 \end{aligned}$$

性质(3)与性质(4)使得相对熵可以用于度量两个分布的相似性。

## 2. 交叉熵

交叉熵，全称为cross - entropy，它的定义为：

$$H(p(x), q(x)) = H(X) + \text{KL}(p(x)||q(x)) \quad (4.54)$$

其中 $H(X)$ 表示随机变量 $X$ 的信息熵， $H(X) = -\sum_{x \in X} p(x) \times \log_2 p(x)$ 。由于真实分布 $p(x)$ 是一个固定值，因此 $H(X)$ 是一个不变量，故有：

$$H(p(x), q(x)) \propto \text{KL}(p(x)||q(x)) \quad (4.55)$$

成立。

化简式(4.54)可以得到：

$$\begin{aligned} H(p(x), q(x)) &= H(X) + \text{KL}(p(x)||q(x)) \\ &= -\sum_{x \in X} p(x) \times \log_2 p(x) + \sum_{x \in X} p(x) \times (\log_2 p(x) - \log_2 q(x)) \\ &= -\sum_{x \in X} p(x) \times \log_2 q(x) \end{aligned} \quad (4.56)$$

式(4.56)比式(4.53)更为简洁，从式(4.55)可以看到交叉熵与相对熵存在一定的等价关系，相对熵的性质对于交叉熵同样适用。因此，一般采用交叉熵来度量两个分布的相似性，特别是在深度学习领域，交叉熵代价函数是常用的目标函数。

## 参考文献：

---

- [1] Stuart Russell. Artificial Intelligence: A Modern Approach (3rd Edition). PE; 3 edition (2015). ISBN-13: 978-9332543515
- [2] ET Jaynes. Probability Theory: The Logic of Science. Cambridge University Press; 1 edition (June 9, 2003). ISBN-13: 978-0521592710
- [3] Thomas M Cover. Elements of Information Theory 2nd Edition. Wiley, 2 edition (2006). ISBN-13: 978-8126541942

# 5

## 概率图模型

人类对未知世界的认知过程，归根结底是一个利用已有的经验知识去预测或者理解不确定性的过程。第4章介绍了概率统计相关的知识，概率统计模型为解决这类不确定性问题提供了一种数学框架，即基于训练样本数据来学习变量间的关系。

考虑这样一个生活中的问题<sup>[1]</sup>，现在有两种疾病：流感和花粉热，分别对应于两个二值随机变量`flu`和`hay_fever`，这两种疾病并不互斥，也就是说一个患者可能只患一种，也有可能同时患两种疾病。考虑三个与疾病相关的因素和症状：季节、鼻窦是否充血和肌肉是否疼痛，它们分别对应于三个随机变量：`season`、`congestion`和`pain`。其中，`season`是一个四值随机变量，而`congestion`和`pain`均是二值随机变量，这样整个概率空间就有64（ $2 \times 2 \times 4 \times 2 \times 2$ ）个值，分别对应于上述5个随机变量的取值。当给定联合概率分布，就可以查询患者患有某一种疾病的概率，比如在秋季中，患者有鼻窦充血，但没有肌肉疼痛，他患流感的概率可以形式化地表示为：

$$p(\text{flu} = \text{true} \mid \text{season} = \text{fall}, \text{congestion} = \text{true}, \text{pain} = \text{false})$$

在上面的例子中，64个值的概率空间也许还可以接受，但对于更高维数据的处理就不理想了，假设现有 $n$ 维随机变量集合 $X = (X_1, X_2, \dots, X_n)$ ，我们的目标是求取 $X$ 的联合概率分布 $P(X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)$ ，但即使 $X_i$ 是一个二值随机变量，除非 $n$ 值非常小，否则联合概率分布的表示方法也是非常难处理的，Koller在*Probabilistic Graphical Models*一书中给出了概率统计模型的几个缺点。

- 从计算的角度：求取联合概率分布的操作成本非常昂贵，并且通常由于数据量

过大，也难以在内存中存储。例如，当 $n = 100$ ，且 $X_i$ 是一个简单的二值随机变量时，则概率分布的表示需要 $(2^{100} - 1)$ 个参数。

- 从认知的角度：由于在高维空间中，每一个具体的随机变量集合的概率值都非常小，人们很难将每一个概率取值与具体的事件一一对应。
- 从统计学的角度来看：要从数据中学习出分布，由于参数数量多，需要规模非常庞大的训练数据才能可靠地估计出这么多的参数。

事实上，在实际应用中，变量之间往往存在很多独立性或近似独立性的假设，也就是说每一个随机变量只和极少数的随机变量相关联。概率图模型（Probabilistic Graphical Model，简称PGM），根据变量之间的独立性假设，为我们提供了解决这类问题的一种机制，PGM以概率论和图论为基础，通过图结构将概率模型可视化，使得我们能够观察到复杂分布中变量之间的关系；同时把概率上的复杂计算过程理解为在图上进行信息传递的过程，从而无须关注太多的复杂表达式。

PGM的体系和框架非常庞大，也是人工智能领域非常重要的一个分支。通常来说，它由表示理论、推断理论和学习理论三大部分构成，本章的结构安排如下。

- 首先介绍机器学习中生成模型和判别模型的概念，它们分别对应于概率统计的联合概率分布和条件概率分布。
- 在5.2节将介绍图论的相关知识点。图论是PGM的重要基础，也是PGM的底层数据结构。
- 5.3节和5.4节讲解PGM的表示理论。5.3节介绍贝叶斯网络，也称有向概率图模型，它使用有向边来构造随机变量之间的依赖关系。5.4节介绍马尔科夫网络表示，也称为无向概率图模型，用无向边来表示随机变量之间的相互影响。
- 5.5节、5.6节和5.7节构成PGM的推断理论。其中5.5节和5.6节是精准推断，我们将分别介绍两种常见的精准推断算法：变量消除和信念传播。5.7节介绍近似推断的方法，在本书中，只介绍MCMC采样的原理。
- 5.8节将简单介绍PGM中常用的参数学习方法。根据模型的结构是否已知，PGM的学习理论可以区分为结构确定的参数学习和结构不确定的参数学习，限于本书的篇幅，只讲解模型结构已知前提下的参数学习方法。
- 最后一节将回顾本章的重要知识点，并对本章没有涉及的其他知识点进行概要的介绍。

## 5.1 生成模型与判别模型

在监督学习中获取的训练数据通常由两部分构成，分别是输入随机变量集合 $X$ 和输出随机变量 $Y$ （本节只考虑单变量输出问题，也就是说随机变量 $Y$ 是一个一维随机变量），其中 $X = (x_1, x_2, \dots, x_n)$ 是一个多维随机变量， $x_i$ 表示第 $i$ 个特征随机变量，机器学习的任务是建立一个模型，并应用这一模型对任意给定的输入 $X$ ，预测输出 $Y$ 。

从形式上，监督学习模型可以分为**概率模型**和**非概率模型**，概率模型是利用训练样本数据，通过学习条件概率分布 $P(Y|X)$ 来进行推断决策，而非概率模型则是通过学习得到决策函数 $Y = f(X)$ 来进行推断。

从算法层面上，监督学习模型又可以分为生成模型和判别模型。

**生成模型**：生成模型的目标是求取联合概率分布 $P(X, Y)$ ，然后由条件概率公式(4.20)求取条件概率分布 $P(Y|X)$ ，即：

$$P(Y|X) = \frac{P(X, Y)}{P(X)} \quad (5.1)$$

典型的生成模型包括：朴素贝叶斯、隐马尔科夫模型等。之所以称式(5.1)为生成模型，是因为模型不但可以用来预测结果输出 $\arg\max_Y(P(Y|X))$ ，还可以通过联合分布 $P(X, Y)$ 来生成新的样本数据集 $(x_i, y_i)$ 。

假设有训练样本数据 $(x, y)$ ：(0,0)、(1,0)、(2,0)、(1,1)、(2,0)、(2,1)。

生成模型的目标是得到二维随机变量 $(x, y)$ 的联合分布律 $P(x, y)$ ，如图5.1所示。

$y \backslash x$	0	1	2
0	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{2}{6}$
1	0	$\frac{1}{6}$	$\frac{1}{6}$

图5.1 二维随机变量的联合分布律

**判别模型**：判别模型是由训练数据直接求取决策函数 $Y = f(X)$ 或条件分布 $P(Y|X)$ 。判别模型并不需要关心 $X$ 和 $Y$ 之间的生成关系，它直接关心的是对于给定的输入 $X$ 应该得到怎么样的输出 $Y$ 。机器学习中大部分的分类模型都属于判别模型，例如，感知机、决策树、支持向量机、条件随机场等。

回顾上面的例子，判别模型的目标是得到二维随机变量 $(x, y)$ 的条件分布律 $P(y|x)$ ，如图5.2所示。

$y \backslash x$			
	0	1	2
0	1	$\frac{1}{2}$	$\frac{2}{3}$
1	0	$\frac{1}{2}$	$\frac{1}{3}$

图5.2 二维随机变量的条件分布律

一般来说，两种模型之间适用于不同条件下的学习问题，生成模型除了可以应用在预测数据外，还可以还原出数据的联合分布函数，因此生成模型的应用领域更广泛。判别模型得到条件概率或决策函数直接用于预测，因此在监督学习中的准确率会更高<sup>[2]</sup>。

## 5.2 图论基础

图结构是很重要的一种数据结构类型，也是概率图模型的两大支柱之一（另一个支柱是概率论），图结构的好处是能够将复杂的分布关系通过可视化的形式来展示。本节简单复习与PGM相关的图论基础知识。

### 5.2.1 图的结构

图 $G = \langle V, E \rangle$ 由两部分类型构成，分别是节点集 $V$ 和边集 $E$ 。

节点集记为 $V = (v_1, v_2, \dots, v_n)$ ，也可以简记为 $V(G)$ ，节点 $v_i$ 表示图 $G$ 中的任意一个节点。

边集记为 $E = (e_1, e_2, \dots, e_n)$ ，简记为 $E(G)$ ，边 $e_i$ 表示图 $G$ 中连接任意两个节点间的曲线。边可以划分为有向边与无向边，有向边记为 $e_{ij} = v_i \rightarrow v_j$ ，表示从节点 $v_i$ 指向节点 $v_j$ ，无向边记为 $e_{ij} = v_i - v_j$ 。

为了方便起见，在本书后面描述边时，统一使用元组 $(v_i, v_j)$ 来表示， $v_i$ 和 $v_j$ 是图中的任意两个节点，对于有向边， $(v_i, v_j) = v_i \rightarrow v_j$ ；对于无向边， $(v_i, v_j) = v_i - v_j$ 。

如果一个图的边都由有向边构成，称为**有向图**，反之，一个图的边都由无向边构成，则称为**无向图**。在本书中，我们讨论的图都是**简单图**，即不存在自环与重边。

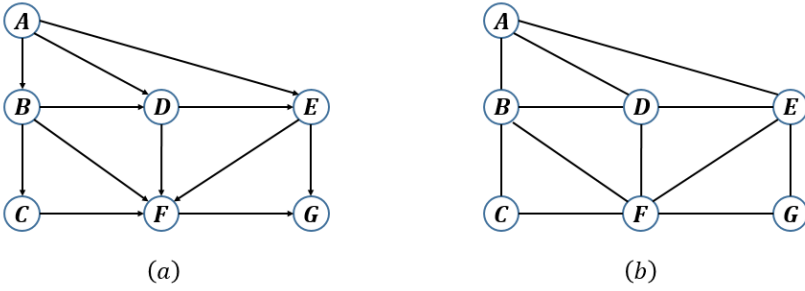
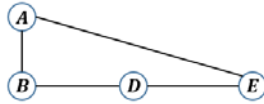


图5.3 (a)有向图模型, (b)无向图模型

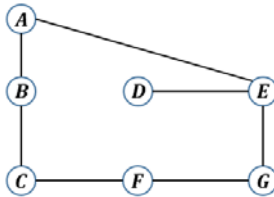
### 5.2.2 子图

很多情况下, 我们只考虑与节点子集相关的部分图。首先了解下面几个与子图相关的重要概念。

(1) **子图**: 给定两个图结构, 记为 $G$ 和 $G^*$ , 如果 $V(G^*)$ 是 $V(G)$ 的子集, 且 $E(G^*)$ 是 $E(G)$ 的子集 (且 $E(G^*)$ 中只能包含将 $V(G^*)$ 中的节点相连的边), 那么称图 $G^*$ 是图 $G$ 的子图。图5.4是图5.3(b)的关于节点集 $\{A, B, D, E\}$ 的一个子图。

图5.4 图5.3(b)的子图 $G^*$ , 其中 $V(G^*) = \{A, B, D, E\}$ ,  $E(G^*) = \{(A, B), (B, D), (D, E), (A, E)\}$ 

(2) **生成子图**: 如果图 $G$ 的子图 $G^*$ 满足 $V(G) = V(G^*)$ , 即图 $G^*$ 包含图 $G$ 的所有节点, 则称图 $G^*$ 是图 $G$ 的支撑子图或生成子图。图5.5是关于图5.3(b)的一个生成子图, 其中边集为 $\{(A, B), (B, C), (A, E), (D, E), (C, F), (G, F), (E, G)\}$ 。

图5.5 图5.3(b)的生成子图 $G^*$ , 其中 $V(G^*) = \{A, B, C, D, E, F, G\}$ ,

$$E(G^*) = \{(A, B), (B, C), (C, F), (F, G), (D, E), (A, E), (E, G)\}$$

(3) **导出子图**: 对于图 $G$ 的子图 $G^*$ , 若 $G^*$ 的任意一条边 $(u, v)$ , 均满足 $(u, v) \in E(G^*)$ ,

当且仅当 $(u, v) \in E(G)$ 时，即子图 $G^*$ 包含了图 $G$ 中所有两个端点都在 $V(G^*)$ 的边，则称 $G^*$ 是 $G$ 的导出子图。图5.6(a)是关于图5.3(b)的一个生成子图，其中节点集合包括 $\{B, C, D, F\}$ ，边集合包括 $\{(B, C), (B, F), (B, D), (C, F), (D, F)\}$ 。但图5.6(b)不是图5.3(b)的一个生成子图，因为图中没有包含边 $(B, F)$ 。

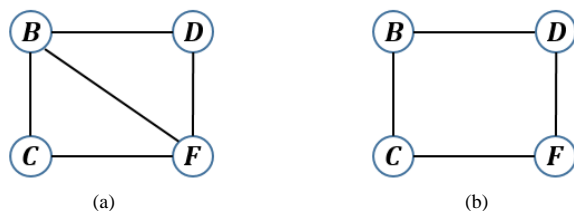


图5.6 图5.6(a)是图5.3(b)的导出子图，其中 $V(G^*) = \{B, C, D, F\}$ ， $E(G^*) = \{(B, C), (C, F), (D, F), (B, D), (B, F)\}$  (b)不是5.3(b)的导出子图，因为 $(B, F) \in E(G)$ ，但 $(B, F) \notin E(G^*)$

(4) **团**：给定两个无向图： $G$ 和 $G^*$ ，其中图 $G^*$ 是图 $G$ 的子图，若图 $G^*$ 的任意两个节点都有边相连，则称子图 $G^*$ 是图 $G$ 的一个团，若一个团中加入另外的一个节点和相应的边后不能再形成团，那么这个团也被称为最大团。任意两个节点 $\{v_i, v_j\}$ ，若该两个节点之间有边相连，那么 $\{v_i, v_j\}$ 必然是一个团。我们观察图5.3(b)，图5.7(a)是其中的一个最大团，但图5.7(b)不构成团，因为 $(C, D)$ 间没有边相连。



图5.7 图5.7(a)是图5.3(b)其中一个最大团，图5.7(b)不是团，因为 $C$ 和 $D$ 间没有边相连

### 5.2.3 路径、迹、环与拓扑排序

**路径 (path)**：对于图 $G$ 中的一个顶点序列 $\{v_0, v_1, \dots, v_k\}$ ，如果任意两个相邻的节点 $v_i$ 与 $v_{i+1}$ ，都有一条有向边 $v_i \rightarrow v_{i+1}$ 或无向边 $v_i - v_{i+1}$ ，那么这个顶点序列就构成了一个长度为 $k$ 的路径。

**迹 (trail)**：对于图 $G$ 中的一个顶点序列 $\{v_0, v_1, \dots, v_k\}$ ，如果任意两个相邻的节点 $v_i$ 与 $v_{i+1}$ 之间都有边相连，那么这个顶点序列就构成了一个长度为 $k$ 的迹。路径和迹的区别：在迹的定义中，对于有向边，不要求边必须是从 $v_i$ 指向 $v_{i+1}$ 。因此，在有向图



中一条路径必然是迹，但反之不成立，一条迹不一定是路径；在无向图中，迹与路径是等价的关系。

例如，在图5.3(a)中， $A \rightarrow B \rightarrow C \rightarrow F \rightarrow G$ 既是一条路径，也是一条迹；而 $A \rightarrow B \rightarrow C \rightarrow F \leftarrow D$ 是一条迹，但不是一条有效路径。

**环 (cycle) :** 如果图 $G$ 存在一条长度为 $k$ 的路径： $v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_k$ ，且 $v_0 = v_k$ ，那么称该路径构成一个环。

**有向无环图 (directed acyclic graph, 简称DAG) :** 若一个有向图中不存在环，则称该有向图是一个有向无环图。DAG是构成贝叶斯网络的基础。

图5.8(a)是一个典型的有向无环图，但图5.8(b)不是有向无环图，因为节点集 $\{C, D, E\}$ 构成一个环。

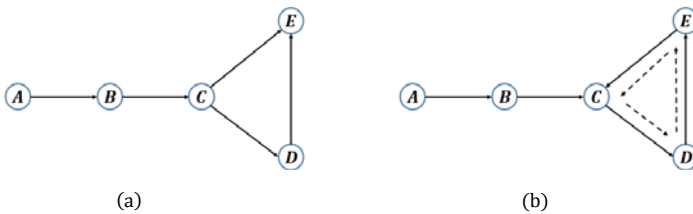


图5.8 (a) 有向无环图, (b) 非有向无环图

**拓扑排序:** 对于一个有向无环图，由于图中节点之间不存在环，因此，节点之间存在先后的次序关系，我们把这种节点先后次序的关系称为拓扑排序。

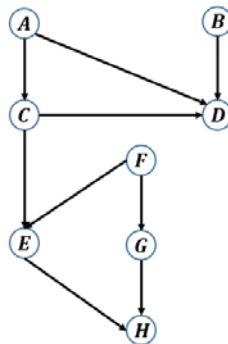


图5.9 DAG图

求取拓扑排序有很多种高效的方法，这里分别给出基于广度优先搜索 (BFS) 的算法，以及基于深度优先搜索 (DFS) 的方法。

基于BFS的求解策略，如图5.10所示。

算法5.1 基于BFS的拓扑排序

1. 标记图G中的所有节点没有被访问

2. 令初始的拓扑序列为空{Null}

3. 计算每一个节点的入度数，把入度为0的节点放到一个队列Q中，并标记节点已经被访问

4. 执行BFS操作，当队列Q不为空的时候，执行第5步；否则执行第6步

5. 当队列Q不为空，循环执行下面的操作

5.1 取出队头元素head，把head插入到拓扑序列中{...,head}

5.2 遍历head的所有相邻节点v

5.2.1 令v的入度减1

5.2.2 若v的入度数变为0，则把v插入到队列Q中，并标记v已被访问

5.2.3 继续遍历下一个邻节点

6. 得到最终的拓扑排序{v<sub>0</sub>,v<sub>1</sub>,...,v<sub>m</sub>}

图5.10 基于BFS的拓扑排序求解算法

利用算法5.1求解图5.9的拓扑排序的过程如图5.11所示。

T	队列	拓扑排序
t <sub>1</sub>	A, B, F	
t <sub>2</sub>	B, F, C	A
t <sub>3</sub>	F, C	A, B
t <sub>4</sub>	C, G	A, B, F
t <sub>5</sub>	G, E, D	A, B, F, C
t <sub>6</sub>	E, D	A, B, F, C, G
t <sub>7</sub>	D, H	A, B, F, C, G, E
t <sub>8</sub>	H	A, B, F, C, G, E, D
t <sub>9</sub>		A, B, F, C, G, E, D, H

图5.11 基于BFS的拓扑排序的求解过程

基于DFS的求解策略，如图5.12所示。

94

算法5.2 基于DFS的拓扑排序

1. 任意选取图 $G$ 中一个入度为0的节点作为根节点 $root$

2. 从根节点 $root$ 开始执行深度遍历

3. 记录每一个节点的开始访问时间 $s$ 和结束访问时间 $e$

4. 按结束时间从最大到最小进行节点排序就是最终的拓扑排序

图5.12 基于DFS的拓扑排序求解算法

利用算法5.2求解图5.9的拓扑排序的过程如图5.13所示，首先选取节点 $F$ 作为根节点，从节点 $F$ 开始执行DFS，得到每个节点的最后访问时间，由最后访问时间从大到小进行排序，得到最终的拓扑排序为： $B, A, C, D, F, E, G, H$ 。

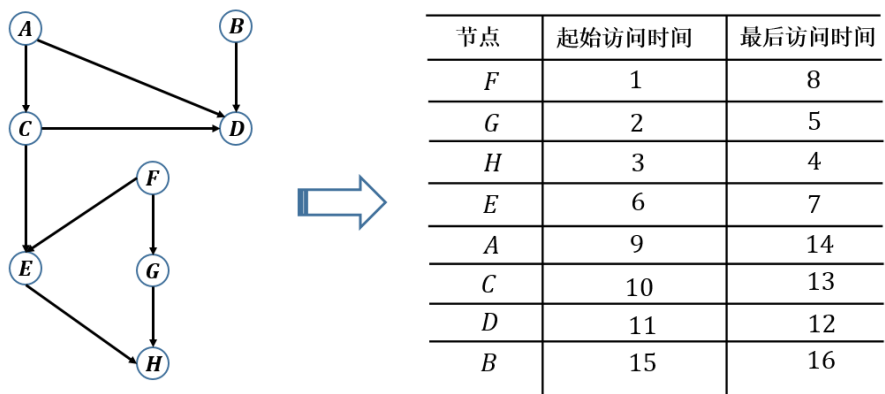


图5.13 基于DFS的拓扑排序的求解过程

从上面两个例子可以看到，对于一个DAG图采用不同的算法策略得到的拓扑排序序列并不唯一。

下面开始讲解PGM的表示理论，利用图来可视化概率分布有两种不同的表示形式：一种是基于有向图的结构，另一种则是基于无向图模型。

5.3 贝叶斯网络

有向概率图模型，也称为贝叶斯网络，利用图的节点来表示随机变量，有向边表示随机变量之间的依赖关系。本节主要从两个方面来学习贝叶斯网络模型表示：第一，如何利用概率分布中的条件独立性，来更紧凑地表示高维分布；第二，如何使用图结

构来可视化这种紧凑表示。

5.3.1 因子分解

本节从三个例子出发，由简单到复杂，让读者先从直观上意识到如何利用概率分布中的条件独立性，来对联合分布进行更紧凑的表示。

1. 例一

首先思考一个生活中的问题，流感往往会伴随着咽喉疼痛的症状，但咽喉疼痛不一定表示患有流感，也有可能是天气干燥引起的喉咙发炎。为了讨论方便，分别用两个随机变量 $x$ 和 $y$ 来表示，其中 $x$ 表示患者是否患有流感， $y$ 表示患者是否有咽喉疼痛，它们均是二值随机变量，因此，可以列出联合概率分布 $p(x,y)$ 如图5.14所示。

$y \backslash x$	0	1
	0	1
0	0.1	0.1
1	0.1	0.7

图5.14 流感和咽喉疼痛的联合概率分布

由第4章的条件概率公式(4.20)可知，上面的联合概率分布存在一个更自然的表示：

$$p(x,y) = p(x)p(y|x) \tag{5.2}$$

其中， $p(x)$ 也被称为先验概率 (prior probability)， $p(y|x)$ 是一个条件概率分布，称为似然性 (likelihood)。上面的公式不只是形式上的改变，从直观上来看，它用一种与因果关系更匹配的方式来表示两者之间的关系；从数学上来看，可以得到如图5.15的另一种 $p(x,y)$ 表示方法。

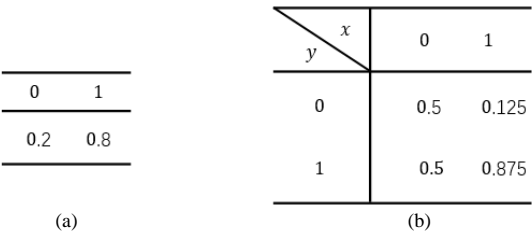


图5.15 (a)图是先验概率 $p(x)$ 的分布，(b)图是似然性 $p(y|x)$ 的分布

考虑将式(5.2)参数化，对于先验概率 $p(x)$ ，由于 $x$ 是一个二值随机变量，可以用

一个参数 $\theta_x$ 来完全表示 $p(x)$ ，规定 $\theta_x = p(x = 1)$ ， $1 - \theta_x = p(x = 0)$ 。同理，对于条件概率 $p(y|x)$ ，用两个参数 $\theta_{y|x_0}$ 和 $\theta_{y|x_1}$ 来分别表示 $p(y|x = 0)$ 和 $p(y|x = 1)$ ，我们约定：

$$\begin{aligned}\theta_{y|x_0} &= p(y = 1|x = 0) & 1 - \theta_{y|x_0} &= p(y = 0|x = 0) \\ \theta_{y|x_1} &= p(y = 1|x = 1) & 1 - \theta_{y|x_1} &= p(y = 0|x = 1)\end{aligned}$$

经过参数化表示后可以把4值的概率空间表示用3个参数来替代，这种表示方法被称为**条件参数化**。

## 2. 例二

在例一中，利用因子分解来表示概率分布，虽然比联合概率分布的表示更加自然，但并没有更紧凑，不过我们马上会发现，这种因子分解为更复杂概率分布的紧凑表示奠定了基础。继续考察例一，现在添加另一个随机变量：季节，记为 $z$ 。这种情况下，概率空间是定义在三个随机变量 $x$ 、 $y$ 、 $z$ 上的联合分布。 $x$ 与 $y$ 表示的含义和前面一样， $z$ 可以取春（0）、夏（1）、秋（2）、冬（3）四个值，这样由 $x$ 、 $y$ 、 $z$ 构成的联合分布的概率空间有 $16(2 \times 2 \times 4)$ 个值。

为了简化讨论，假设季节与咽喉疼痛在患有流感的条件下是相互独立的，记为 $(z \perp y)|x$ ，进一步有：

$$p(x, y, z) = p((y, z)|x) \times p(x) = p(x)p(y|x)p(z|x) \quad (5.3)$$

首先仿照图5.15，给出式(5.3)的条件概率分布，如图5.16所示。

	$x$		$z$			
	$y$		$x$			
	0	1	0	1	2	4
0	0.2	0.8	0.22	0.25	0.28	0.25
1			0.35	0.12	0.05	0.48

(a)
(b)
(c)

图5.16 (a)图 $p(x)$ 的概率分布，(b)图 $p(y|x)$ 的概率分布，(c)图 $p(z|x)$ 的概率分布

考察 $p(z|x)$ 的参数化表示，我们用两个参数向量 $\theta_{z|x_0}$ 和 $\theta_{z|x_1}$ 来分别表示 $p(z|x = 0)$ 和 $p(z|x = 1)$ ，由于 $z$ 是一个四值随机变量，参数向量 $\theta_{z|x_0}$ 和 $\theta_{z|x_1}$ 都需要三个参数来表示，其中：

$$\theta_{z|x_0} = (\theta_{z_0|x_0}, \theta_{z_1|x_0}, \theta_{z_2|x_0})^T \quad \theta_{z|x_1} = (\theta_{z_0|x_1}, \theta_{z_1|x_1}, \theta_{z_2|x_1})^T$$

具体来说，每一个取值分别为：

$$\theta_{z_0|x_0} = p(z = 0|x = 0) \quad \theta_{z_1|x_0} = p(z = 1|x = 0) \quad \theta_{z_2|x_0} = p(z = 2|x = 0)$$

$$1 - \theta_{z_0|x_0} - \theta_{z_1|x_0} - \theta_{z_2|x_0} = p(z = 3|x = 0)$$

$$\theta_{z_0|x_1} = p(z = 0|x = 1) \quad \theta_{z_1|x_1} = p(z = 1|x = 1) \quad \theta_{z_2|x_1} = p(z = 2|x = 1)$$

$$1 - \theta_{z_0|x_1} - \theta_{z_1|x_1} - \theta_{z_2|x_1} = p(z = 3|x = 1)$$

这样式(5.3)的条件参数表示一共需要9个参数，如果使用联合概率分布表示，其概率空间大小为16，需要用15个独立参数，这里能对比出因子分解的条件参数化表示比联合概率分布表示更紧凑，所需要的参数更少。

除此之外，因子分解表示的另一个优势是模块性，也就是说当添加了一个新的随机变量，联合概率分布会在整体上发生改变，比如添加了新变量 $z$ ，需要从原来的4个值修改为全新的16个值，但在因子分解表示中，对比式(5.2)与式(5.3)，只需要在末尾添加新的关系 $p(z|x)$ 即可。这个模块性对于解决实际问题时，遇到的更复杂的关系表示尤其方便。

### 3. 例三

朴素贝叶斯是机器学习中常用的分类模型。对于训练数据集 $(X, C)$ ，其中 $C$ 为类别， $X = (X_1, X_2, \dots, X_n)$ 表示特征向量，朴素贝叶斯假设在给定某一个类别 $C$ 的条件下，各特征之间是相互独立的，即满足：

$$(X_i \perp X_j) | C \quad i, j \in \{1, 2, \dots, n\}, \quad i \neq j$$

基于这种条件独立性，我们把联合概率分布 $p(C, X_1, X_2, \dots, X_n)$ 化简为：

$$p(C, X_1, X_2, \dots, X_n) = p(C) \prod_{i=1}^n p(X_i | C) \quad (5.4)$$

分析式(5.4)的因子分解表示。如果用联合概率分布来表示，设 $C$ 和 $X_i$ 均为二值随机变量，那么需要 $(2^{n+1} - 1)$ 个参数来表示，但使用因子分解来表示，只需要 $(2 \times n + 1)$ 个参数，参数空间与变量之间是一个线性关系。

从上面的三个例子中，我们不难发现，使概率分布能更紧凑表示的前提是：**不同的随机变量集合之间存在独立关系**。这种独立关系能够简化联合概率分布的表示，从

而用更少的参数就能表达出复杂的分布。那么如何表示这种独立关系呢？当随机变量很多，相互间的独立关系错综复杂的时候，我们不可能把所有的独立性都列出来。概率图模型的表示理论就是解决概率分布中独立性表示的问题。本节首先讲解有向图表示，在5.4节将继续讲解无向图表示。

### 5.3.2 局部马尔科夫独立性断言

贝叶斯网络本质上是一个有向无环图（Directed Acyclic Graph，简称DAG），通常记为 $G$ ，它的每一个节点 $X_i$ 对应于一个随机变量，有向边 $X_i \rightarrow X_j$ 表示随机变量 $X_i$ 对随机变量 $X_j$ 的影响。图5.17展示了一个典型的贝叶斯网络。首先来定义下面的变量表示。

- 记 $Pa_{X_i}^G$ 表示节点 $X_i$ 在 $G$ 中的所有父节点集合。
- $NonDescendants_{X_i}$ 表示节点 $X_i$ 在 $G$ 中的所有非后代节点集合（不包括父节点）。

**定义5.1:** 对于任意给定的贝叶斯网络 $G$ ，它存在下面的**局部马尔科夫独立性断言**，即满足对于每一个随机变量 $X_i$ ，有 $(X_i \perp NonDescendants_{X_i} | Pa_{X_i}^G)$ 成立。贝叶斯网络的局部马尔科夫独立性，简称为局部独立性。

局部独立性断言指出，对于任意的随机变量 $X_i$ ，在给定父节点集合的条件下， $X_i$ 与其非后代节点条件独立。由贝叶斯网络 $G$ 的局部独立性断言所导出的所有条件独立性集合，记为 $I_l(G)$ 。

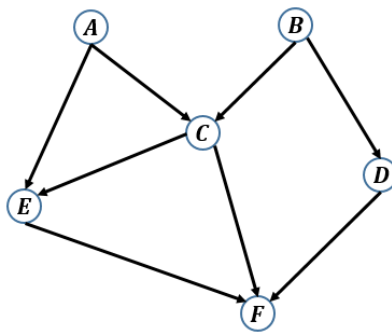


图5.17 贝叶斯网络示例

考察如图5.17所示的贝叶斯网络，对于随机变量 $C$ ，其父节点为 $Pa_C^G = \{A, B\}$ ，非后代节点为 $NonDescendants_C = \{D\}$ 。由局部马尔科夫独立性，有 $(C \perp D) | \{A, B\} \Leftrightarrow p(C|A, B) = p(D|A, B)$ 成立。读者也可以分别自行考察每一个随机变量。对图5.17执

行相同的操作，最后得到的所有条件独立集合，就构成了 $I_l(G)$ ，这里不再详述。

5.3.3 I-Map与因子分解

对于定义在随机变量集合 $X = (X_1, X_2, \dots, X_n)$ 的分布 $P$ ， $I(P)$ 为定义在 $P$ 中的由形如 $(x \perp y|z)$ 构成的所有条件独立集合。若由 $(X_1, X_2, \dots, X_n)$ 构成的贝叶斯网络 $G$ ，满足 $I_l(G) \subseteq I(P)$ ，即 $P$ 满足与 $G$ 相关的局部独立性，那么称 $G$ 是 $P$ 的一个**独立图**，简称**I-Map**。

通过一个具体的例子来考察I-Map的含义。假设现有二值随机变量 $A$ 和 $B$ ，由 $A$ 和 $B$ 构成的所有可能的贝叶斯网络如图5.18所示，分别是随机变量 $A$ 与随机变量 $B$ ，两者之间没有边相连，记为 $G_\emptyset$ ；有一条有向边从随机变量 $A$ 指向随机变量 $B$ ，记为 $G_{A \rightarrow B}$ ；有一条有向边从随机变量 $B$ 指向随机变量 $A$ ，记为 $G_{B \rightarrow A}$ 。

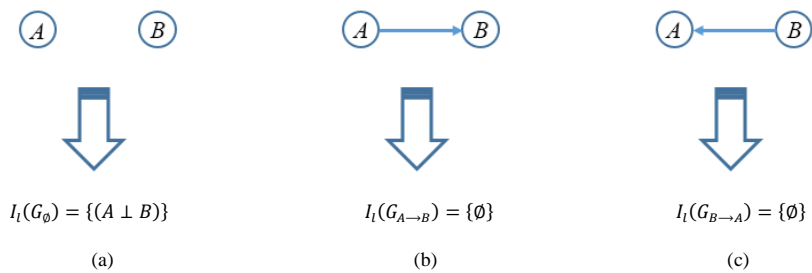


图5.18 由两个随机变量 $A$ 和 $B$ 构成的所有可能的贝叶斯网络，并由贝叶斯网络的局部独立性断言得到相应的条件独立性集合 $I_l(G)$

定义一个在二值随机变量 $A$ 和 $B$ 上的概率分布 $P$ ，其联合分布律如图5.19所示。

$A$	$B$	$P(A, B)$
0	0	0.08
0	1	0.32
1	0	0.12
1	1	0.48

图5.19 概率分布 $P$ 的联合分布

考察联合概率分布 $P(A, B)$ ，观察随机变量 $A$ 和 $B$ 的不同取值对应的 $P$ 值变化情况。

$P(A = 0) = 0.4, P(B = 0) = 0.2 \Rightarrow P(A = 0) \times P(B = 0) = P(A = 0, B = 0) = 0.08$



$$P(A = 0) = 0.4, P(B = 1) = 0.8 \Rightarrow P(A = 0) \times P(B = 1) = P(A = 0, B = 1) = 0.32$$

$$P(A = 1) = 0.6, P(B = 0) = 0.2 \Rightarrow P(A = 1) \times P(B = 0) = P(A = 1, B = 0) = 0.12$$

$$P(A = 1) = 0.6, P(B = 1) = 0.8 \Rightarrow P(A = 1) \times P(B = 1) = P(A = 1, B = 1) = 0.48$$

故有 $P(A) \times P(B) = P(A, B)$ 成立, 因此 $I(P) = (A \perp B)$ 。图5.18定义三个贝叶斯网络, 对应的局部独立性集合分别为:  $I_l(G_\emptyset) = \{(A \perp B)\}$ 、 $I_l(G_{A \rightarrow B}) = \{\emptyset\}$ 、 $I_l(G_{B \rightarrow A}) = \{\emptyset\}$ , 可以看到它们均为 $I(P)$ 的子集, 因此, 图5.18定义三个贝叶斯网络都是 $P$ 的I-Map。

### 1. 从I-Map到因子分解

首先考察一个具体的例子, 然后给出如何由贝叶斯网络来简化概率分布的因子分解表示。

图5.17是定义在随机变量集合 $\{A, B, C, D, E, F\}$ 上的贝叶斯网络, 记为 $G$ , 计算随机变量集合 $\{A, B, C, D, E, F\}$ 上的联合概率分布, 由概率公式的链式法则得:

$$P(A, B, C, D, E, F) =$$

$$P(A)P(B|A)P(C|A, B)P(D|A, B, C)P(E|A, B, C, D)P(F|A, B, C, D, E) \quad (5.5)$$

若贝叶斯网络 $G$ 是分布 $P$ 上的一个I-Map, 根据贝叶斯网络的局部独立性断言, 有下面等式成立:

$$A \perp B \Rightarrow P(B|A) = P(B) \quad (5.6)$$

$$D \perp A, C, E|B \Rightarrow P(D|A, B, C) = P(D|B) \quad (5.7)$$

$$E \perp D, B|\{A, C\} \Rightarrow P(E|A, B, C, D) = P(E|A, C) \quad (5.8)$$

$$F \perp A, B|\{E, C, D\} \Rightarrow P(F|A, B, C, D, E) = P(F|C, D, E) \quad (5.9)$$

把式(5.6)至式(5.9)代入式(5.5), 可将联合概率分布化简为:

$$P(A, B, C, D, E, F) =$$

$$P(A)P(B|A)P(C|A, B)P(D|A, B, C)P(E|A, B, C, D)P(F|A, B, C, D, E) \Leftrightarrow$$

$$P(A, B, C, D, E, F) = P(A)P(B)P(C|A, B)P(D|B)P(E|A, C)P(F|C, D, E) \quad (5.10)$$

前面的例子表明, 利用贝叶斯网络蕴含的独立性可以将一个复杂的联合概率分布

化简为简单的因子乘积。下面给出由贝叶斯网络所表示的概率分布的形式化定义。

**定义5.2：**令 $G$ 是定义在随机变量集合 $X_1, X_2, \dots, X_n$ 上的贝叶斯网络， $P$ 是定义在随机变量 $X_1, X_2, \dots, X_n$ 上的概率分布，且满足 $I_l(G) \subseteq I(P)$ ，则随机变量 $X_1, X_2, \dots, X_n$ 的联合概率分布可以表示为乘积：

$$P(X_1, X_2, \dots, X_n) = \prod_{i=1}^n P(X_i | Pa_{X_i}^G) \quad (5.11)$$

式(5.11)被称为 $P$ 的**因子分解**，也称为**贝叶斯链式法则**。单个因子 $P(X_i | Pa_{X_i}^G)$ 称为**条件概率分布 (Conditional Probability Distribution, 简称CPD)**或**局部概率模型**。

证明：对随机变量集合 $\{X_1, X_2, \dots, X_n\}$ ，可以得到常规的联合概率分布

$$P(X_1, X_2, \dots, X_n) = \prod_{i=1}^n P(X_i | X_1, X_2, \dots, X_{i-1}) \quad (5.12)$$

不失一般性，设 $X_1, X_2, \dots, X_n$ 恰好是图 $G$ 的一个拓扑排序，那么对于式(5.12)的任意一个因子 $P(X_i | X_1, X_2, \dots, X_{i-1})$ ，因 $X_1, X_2, \dots, X_i$ 是按拓扑排序的顺序出现，故 $X_i$ 的父节点都在 $X_1, X_2, \dots, X_{i-1}$ 中，且 $X_1, X_2, \dots, X_{i-1}$ 不含有 $X_i$ 的后代节点，故满足：

$$\{X_1, X_2, \dots, X_{i-1}\} = \{Pa_{X_i}^G \cup S\} \quad (5.13)$$

其中 $S \subseteq \text{NonDescendants}_{X_i}$ ，由局部独立性可得：

$$X_i \perp S | Pa_{X_i}^G \quad (5.14)$$

从而有：

$$P(X_i | X_1, X_2, \dots, X_{i-1}) = P(X_i | Pa_{X_i}^G) \quad (5.15)$$

成立。由 $P(X_i | X_1, X_2, \dots, X_{i-1})$ 的任意性，得式(5.11)成立，证毕。

## 2. 从因子分解到I-Map

**定义5.3：**令 $G$ 是定义在随机变量 $X_1, X_2, \dots, X_n$ 上的贝叶斯网络， $P$ 是定义在随机变量 $X_1, X_2, \dots, X_n$ 上的概率分布，若 $P$ 根据网络 $G$ 来进行因子分解，也就是说随机变量 $X_1, X_2, \dots, X_n$ 的联合概率分布可以表示为下面的乘积：

$$P(X_1, X_2, \dots, X_n) = \prod_{i=1}^n P(X_i | Pa_{X_i}^G) \quad (5.16)$$

那么必有  $I_l(G) \subseteq I(P)$  成立，则  $G$  是  $P$  的一个 I-Map。

证明：不失一般性，设  $X_1, X_2, \dots, X_n$  是图  $G$  的一个拓扑排序，对于式(5.12)的任意因子  $P(X_i | X_1, X_2, \dots, X_{i-1})$ ，有：

$$P(X_i | X_1, X_2, \dots, X_{i-1}) = \frac{P(X_1, X_2, \dots, X_{i-1}, X_i)}{P(X_1, X_2, \dots, X_{i-1})} \quad (5.17)$$

把式(5.16)代入式(5.17)，有：

$$P(X_i | X_1, X_2, \dots, X_{i-1}) = P(X_i | Pa_{X_i}^G) \quad (5.18)$$

式(5.18)表明，在给定  $X_i$  的父节点  $Pa_{X_i}^G$  的条件下， $X_i$  与  $\{X_1, X_2, \dots, X_{i-1}\}$  中所有的非后代节点  $\text{NonDescendants}_{X_i}$  条件独立，即  $X_i$  满足贝叶斯网络的局部独立性断言，即满足：

$$(X_i \perp \text{NonDescendants}_{X_i} | Pa_{X_i}^G) \in I(P)$$

由  $i$  的任意性可得  $I_l(G) \subseteq I(P)$  成立。证毕。

### 5.3.4 有效迹

在5.3.3节讨论的独立性都是条件独立性，也就是说，在给定某些观察条件的情况下，随机变量之间存在独立性。本节给出贝叶斯网络所蕴含的更一般的独立性定义。

假设贝叶斯网络  $G$  的顶点集合为  $\{X_1, X_2, \dots, X_n\}$ ， $X_i$  与  $X_j$  相互独立，是指  $X_i$  的取值不会影响  $X_j$  的取值，反之亦然。为了讨论任意的节点  $X_i$  与  $X_j$  的独立性，可以来考察其逆问题，也就是什么情况下  $X_i$  能影响  $X_j$ 。如果  $X_i$  能影响  $X_j$ ，那么  $X_i \perp X_j$  自然不成立。

为了后面的讨论方便，首先给出一个例子，该例子会被用来解析后面提到的各种结论，从而方便读者去理解。图5.20所示的贝叶斯网络由  $I$ 、 $D$ 、 $S$ 、 $G$ 、 $L$  五个随机变量构成，具体的含义如下。

- $I$ ：代表智商Intelligence，是一个二值随机变量，可以取低智商（0）和高智商（1）两个值。
- $D$ ：代表科目的困难程度Difficulty，是一个二值随机变量，可取容易（0）和困难（1）两个值。

- **S**: 代表SAT的分数, 是一个二值随机变量, 可取低分(0)和高分(1)两个值。  
SAT的分数受到智商的影响, 一般来说, 智商越高, 获得高分的概率也越大。
- **G**: 代表科目的分数Grade, 是一个二值随机变量, 可取低分(0)和高分(1)两个值。科目的分数由智商和科目的难度共同影响, 如果一个学生的智商较高, 并且科目的难度比较低, 那么他在该科目获得高分的概率自然会比较高。
- **L**: 代表是否被推荐, 是一个二值随机变量, 可取不被推荐(0)和被推荐(1)两个值。是否被推荐受到科目分数的影响, 通常情况下, 分数越高推荐的概率自然越大。

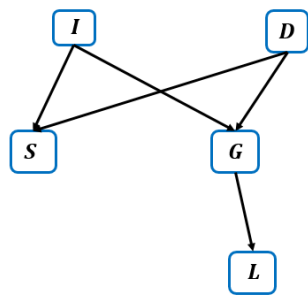


图5.20 贝叶斯网络

考察独立性的逆问题, 也就是在一个贝叶斯网络中, 什么情况下任意两个节点 $X_i$ 与 $X_j$ 之间可以有相互影响? 由于贝叶斯网络是一个弱连通图, 故任意两个节点 $X_i$ 与 $X_j$ 之间必然存在一条迹。两个节点之间的连通关系可以分为直接相连和间接相连。下面分别来分析不同的连通关系对独立性的影响。

**直接相连:** 如果 $X_i$ 直接与 $X_j$ 相连, 那么不管什么情况下, 显然 $X_i$ 与 $X_j$ 都是不可能相互独立的, 如图5.21所示。



图5.21  $X_i$ 与 $X_j$ 直接相连

**间接相连:** 当 $X_i$ 与 $X_j$ 不直接相连, 那么它们之间必存在一条迹, 即满足 $X_i \rightleftharpoons \dots \rightleftharpoons Z \rightleftharpoons \dots \rightleftharpoons X_j$ 。

为了后面讨论的方便, 且不失一般性, 我们只考虑三个节点 $X_i \rightleftharpoons Z \rightleftharpoons X_j$ 的情况。间接关系一共有下面四种可能情况。

## (1) 因果关系。

因果间接关系如图5.22所示,即 $X_i$ 通过 $Z$ 作用于 $X_j$ ,从哲学的角度来看,相当于 $X_i$ 是因,而 $X_j$ 是结果,故而称这种连接关系为因果关系。

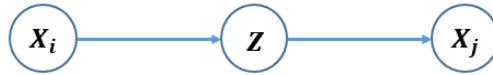


图5.22 因果间接关系

当 $Z$ 没有被观察到时, $X_i$ 的取值能影响到 $X_j$ 的取值。以图5.20的贝叶斯网络为例,图中的 $I \rightarrow G \rightarrow L$ 就是这样一个因果关系,从直观上看,如果不知道分数 $G$ 的取值,但知道一个学生的智商比较高(即 $I$ 为高智商的概率比较大),那么他的分数( $G$ )是高分的概率就比较高,从而被推荐的概率也会越大,即满足: $p(L = 1 | I = 1) \geq p(L = 0 | I = 1)$ 。

当 $Z$ 被观察到时, $X_i$ 的取值不能影响到 $X_j$ 的取值。由贝叶斯网络的局部独立性断言,我们知道在给定分数 $G$ 的条件下,智商 $I$ 与是否被推荐 $L$ 之间条件独立。

## (2) 证据关系。

证据间接关系如图5.23所示,与因果关系相反, $X_j$ 通过 $Z$ 作用于 $X_i$ ,这时候 $X_i$ 是结果,而 $X_j$ 是原因,知道结果的情况下来反推原因,故而称这种连接关系为证据关系。

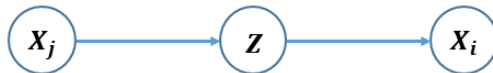


图5.23 证据间接关系

当 $Z$ 没有被观察到时, $X_i$ 的取值能影响到 $X_j$ 的取值。同样以图5.20的贝叶斯网络为例,对于图中的迹 $I \rightarrow G \rightarrow L$ ,从直观上看,如果分数 $G$ 不知道,但知道一个学生被推荐了,那么很自然地认为,该学生是因为这次考试的分数高所以才被推荐,考试分数高,那么我们认为他的智商比较高的概率也比较大,即满足: $p(I = 1 | L = 1) \geq p(I = 0 | L = 1)$ 。

当 $Z$ 被观察到时, $X_i$ 的取值不能影响到 $X_j$ 的取值。同样由贝叶斯网络的局部独立性断言,我们知道在给定分数 $G$ 的条件下,智商 $I$ 与是否被推荐 $L$ 之间条件独立。

## (3) 共同的原因。

共同的原因间接关系如图5.24所示， $Z$ 是 $X_i$ 和 $X_j$ 的共同父节点，故而称这种连接关系为共同原因关系。

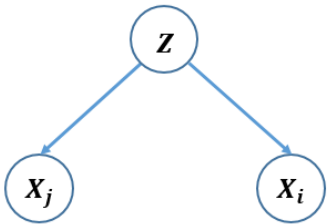


图5.24 共同原因间接关系

当 $Z$ 没有被观察到时， $X_i$ 的取值能影响到 $X_j$ 的取值。观察图5.20，图中的迹  $S \leftarrow I \rightarrow G$  就是属于共同原因的间接关系。从直观上看，如果不知道一个学生的智商  $I$ ，那么 $S$ 与 $G$ 是相互影响的，因为 $S$ 和 $G$ 都是考试分数。如果一个考生的SAT成绩很高，那么我们有理由相信该考生的智商高的概率比较大，当智商比较高时，其他科目的分数获得高分的概率也自然比较大，即满足： $p(G = 1 \mid S = 1) \geq p(G = 0 \mid S = 1)$ 。

当 $Z$ 被观察到时， $X_i$ 的取值不能影响到 $X_j$ 的取值。同样由贝叶斯网络的局部独立性断言，我们知道在给定考生的智商 $I$ 的条件下， $S$ 与 $G$ 之间条件独立。

（4）共同的结果。

共同结果间接关系如图5.25所示， $Z$ 是 $X_i$ 和 $X_j$ 的共同子节点，或者称 $X_i$ 和 $X_j$ 都是 $Z$ 的父节点，故而称这种连接关系为共同结果关系。这种关系图非常类似一个字母 $v$ ，因此也把满足共同结果关系的迹称为 $v$ -结构。这种结构也是4种间接连接关系中最复杂的一种。

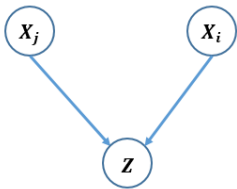


图5.25 共同结果间接关系

当 $Z$ 没有被观察到时， $X_i$ 的取值不能影响到 $X_j$ 的取值。观察图5.20，图中的迹  $I \rightarrow G \leftarrow D$  就是属于共同结果的间接关系，当 $G$ 没有被观察到时，它满足贝叶斯网络的局部独立性断言，故 $I$ 与 $D$ 是相互独立。

当 $Z$ 被观察到时,  $X_i$ 的取值能影响到 $X_j$ 的取值。对于图中的迹 $I \rightarrow G \leftarrow D$ , 当知道考生的科目分数 $G$ 时, 从直观上看, 如果分数 $G$ 很低, 那么判断该考生的智商 $I$ 比较低的概率会较高, 但是如果同时知道科目的难度 $D$ 为很困难时, 那么低智商的概率会降低, 因为如果科目很难, 即使高智商的考生也不一定能取得高分, 即满足:  $p(I = 1 | G = 1, D = 1) \geq p(I = 0 | G = 1, D = 1)$ 。

即使不知道考生的科目分数 $G$ , 但若知道考生是否被推荐 $L$ 时, 由于 $L$ 的取值能影响 $G$ 的取值, 也就是如果一个考生被推荐, 那么他的分数应该比较高, 进一步地, 如果影响到分数, 自然也影响到对智商和科目困难度的判断, 即满足:  $p(I = 1 | L = 1, D = 1) \geq p(I = 0 | L = 1, D = 1)$ 。

下面首先给出有效迹的概念: 我们说一个迹 $X_1 \rightleftharpoons \dots \rightleftharpoons X_i \rightleftharpoons \dots \rightleftharpoons X_n$ 是**有效的**, 当且仅当迹中的任意两个顶点能相互影响。形象地说, **有效迹**相当于一条水管, 水能在管道内任意流动, 没有堵塞。

对于形如 $X_i \rightleftharpoons Z \rightleftharpoons X_j$ 这种只有3个节点的迹, 它是有效迹的充要条件如下。

- 对于因果间接关系迹 $X_i \rightarrow Z \rightarrow X_j$ , 它是有效迹当且仅当没有观察到 $Z$ 。
- 对于证据间接关系迹 $X_j \rightarrow Z \rightarrow X_i$ , 它是有效迹当且仅当没有观察到 $Z$ 。
- 对于共同原因间接关系迹 $X_j \leftarrow Z \rightarrow X_i$ , 它是有效迹当且仅当没有观察到 $Z$ 。
- 对于共同结果间接关系迹 $X_j \rightarrow Z \leftarrow X_i$ , 它是有效迹当且仅当观察到 $Z$ 或者观察到 $Z$ 的至少一个后代。

这个结论推广到一般迹 $X_1 \rightleftharpoons \dots \rightleftharpoons X_i \rightleftharpoons \dots \rightleftharpoons X_n$ , 给出判断迹是有效迹的一般性条件。

**定义5.4:** 设 $G$ 是一个贝叶斯网络, 且 $X_1 \rightleftharpoons \dots \rightleftharpoons X_i \rightleftharpoons \dots \rightleftharpoons X_n$ 是图 $G$ 的一条迹, 令 $Z$ 是观察变量集合, 当在给定 $Z$ 的条件下, 若下面两个条件同时成立, 那么 $X_1 \rightleftharpoons \dots \rightleftharpoons X_i \rightleftharpoons \dots \rightleftharpoons X_n$ 就是图 $G$ 的一条有效迹。

- 若迹中含有 $v$ 结构, 即存在共同结果的间接关系 $X_{i-1} \rightarrow X_i \leftarrow X_{i+1}$ ,  $X_i$ 或者 $X_i$ 的至少一个后代在观察集合 $Z$ 中。
- 对于迹中不属于 $v$ 结构的其他节点 $X_k$ 都不在 $Z$ 中。

由定义5.4, 我们知道对于同一条迹, 不同的观察集合, 或者说在不同的条件下, 可能结果并不相同, 下面考察图5.20的迹 $D \rightarrow G \leftarrow I \rightarrow S$ 。

- 当给定观察  $Z = \emptyset$  时，也就是没有任何条件的前提下，由于迹中存在  $v$  结构  $D \rightarrow G \leftarrow I$ ，但观察集合  $Z = \emptyset$  没有包含  $G$  或其任意一个后代，故  $D \rightarrow G \leftarrow I \rightarrow S$  不是有效迹。事实上，由局部独立性易知，在没有任何观察数据的前提下， $D$  与  $I$  是相互独立的。
- 当给定观察  $Z = \{L\}$  时，对于迹中存在的  $v$  结构  $D \rightarrow G \leftarrow I$ ，观察集合  $Z = \{L\}$  中包含了  $G$  的一个后代  $L$ ，而迹上的其他节点  $\{D, I, S\}$  均不在观察集合  $Z = \{L\}$  中，故  $D \rightarrow G \leftarrow I \rightarrow S$  是有效迹。
- 当给定观察  $Z = \{L, I\}$  时，对于迹中存在的  $v$  结构  $D \rightarrow G \leftarrow I$ ，观察集合  $Z = \{L, I\}$  中包含了  $G$  的一个后代  $L$ ，但观察集合  $Z = \{L, I\}$  同时包含了节点  $I$ ，故  $D \rightarrow G \leftarrow I \rightarrow S$  不是有效迹。事实上，由贝叶斯网络的局部独立性，易知  $(G \perp S | I)$  成立。

### 5.3.5 D-分离与全局马尔科夫独立性

5.3.2节给出了局部独立性的定义，局部独立性把关注点集中在局部的随机变量集合上。本节给出刻画整个贝叶斯网络的全局独立性断言。全局独立性断言依赖于上一节的有效迹的定义和存在的充要条件，下面首先由有效迹导出D-分离 (D-Separation) 的概念。

**定义5.5:** 令  $X$ 、 $Y$ 、 $Z$  是贝叶斯网络的3个节点集，在给定  $Z$  的前提条件下，假如对任意节点  $x \in X$  和  $y \in Y$  之间不存在有效迹，即  $x \rightleftharpoons \cdots \rightleftharpoons Z \rightleftharpoons \cdots \rightleftharpoons y$  不满足定义5.4的有效性条件，那么称  $X$  与  $Y$  在给定  $Z$  的前提条件下是**D-分离**的，记为  $\text{d-sep}_G(X; Y | Z)$ 。

与局部马尔科夫独立性对应，由D-分离定义的独立性集合记为：

$$I(G) = \{(x \perp y | Z) : \text{d-sep}_G(X; Y | Z), x \in X, y \in Y\} \quad (5.19)$$

我们称  $I(G)$  为贝叶斯网络  $G$  所蕴含的**全局马尔科夫独立性集合**，并且  $I(G)$  与  $I_l(G)$  是相互包含的关系，即  $I(G)$  与  $I_l(G)$  本质上是等价的。

## 5.4 马尔科夫网络

首先考虑一个独立性问题：现有四个随机变量  $A$ 、 $B$ 、 $C$ 、 $D$ ，我们希望构建一个概率图模型，使得它满足  $(A \perp C | \{B, D\})$  和  $(B \perp D | \{A, C\})$  这两个独立性条件。贝叶斯



网络是一个有向无环图,要求为每一条边指定方向,用来确定随机变量间的依赖关系,但在上面的问题中,随机变量之间相互影响是对称的, $B$ 、 $D$ 能影响 $A$ 、 $C$ ,同理 $A$ 、 $C$ 也能影响 $B$ 、 $D$ 。因此很难用有向的贝叶斯网络来表示。

从直观上看,无向图模型,由于边的无向性使得我们很自然地认为,它是适合构建这种随机变量间相互影响的模型。本节讨论的重点正是这种无向概率图模型,也称为马尔科夫网络。与贝叶斯网络类似,无向图的每一个节点都对应于一个随机变量,而边则描述了两个节点间相互影响关系。在后面的讨论中,为了更好地区分有向图模型和无向图模型,本节讨论的无向图模型统一采用符号 $H$ 来表示。

### 5.4.1 势函数因子与参数化表示

回顾贝叶斯网络的因子分解表示:贝叶斯网络蕴含局部独立性约束 $I_l(G)$ ,利用 $I_l(G)$ 与贝叶斯链式法则,将联合概率分布表示成因子分解的形式,每一个因子是一个条件概率分布。但对于无向图 $H$ 来说,由无向边连接的两个随机变量,它们之间的影响是等价的,因此不存在一个变量决定另外一个变量的概念,随机变量的联合分布无法表示成CPD相乘的因子分解形式。

无向图中两个节点间的相互影响更像是一种函数关系。也就是说对于随机变量集合 $D$ ,定义一个实值函数 $f: D \rightarrow \mathbf{R}$ 。函数 $f$ 把 $\text{Val}(D)$ (我们用 $\text{Val}(D)$ 表示变量集 $D$ 包含的所有随机变量集合)映射到实数域 $\mathbf{R}$ , $\mathbf{R}$ 的大小决定了 $D$ 中各随机变量之间的某种函数关系。 $\mathbf{R}$ 值越大,表示 $D$ 中各随机变量之间越亲密。把随机变量集合 $D$ 与函数 $f$ 关联起来,这个函数称为一个**无向图因子**。下面形式化地给出马尔科夫网络的因子定义。

**定义5.6:**假设 $X = (X_1, X_2, \dots, X_n)$ 表示随机变量集合,在无向图模型 $H$ 中,因子 $\phi$ 定义为从 $\text{Val}(X)$ 映射到实数域 $\mathbf{R}$ 的一个函数 $f: \text{Val}(X) \rightarrow \mathbf{R}$ ,函数 $f$ 也称为**势函数**(potential functions)。把因子 $\phi$ 包含的变量集合称为 $\phi$ 的**辖域**,记为 $\text{scope}(\phi)$ 。

在无向概率图模型中,因子与势函数是等价的概念。注意,“因子”在贝叶斯网络和马尔科夫网络中所表示的本质是一样的,都定义了局部随机变量集合之间的关系影响,不同的是,在贝叶斯网络中,因子用条件概率分布来定义,即 $\phi = P(X_i | X_1, X_2, \dots, X_{i-1})$ 。

在马尔科夫网络中,因子用势函数来定义。势函数的定义方法有很多,我们将在第11章的玻尔兹曼机中介绍如何利用**能量模型**来定义势函数。此外,在后面的讨论中,

除非有特别的说明，否则都要求势函数的取值大于等于0，也就是说**势函数是一个非负函数**。

前面已经提到，与贝叶斯网络的条件概率分布一样，势函数定义了局部随机变量集合之间的相互影响，为了定义全局影响，需要将局部影响组合起来，在贝叶斯网络中，这些局部的因子是通过相乘的形式进行组合，因此，我们也期望在马尔科夫网络中，联合概率分布也可以表示成势函数相乘的形式，即：

$$P(X) = \phi_1 \phi_2 \dots \phi_n \quad (5.20)$$

但式(5.20)并不是一个合适的因子分解，具体来说，它需要解决下面的两个问题。

首先，式(5.20)定义的概率分布并不能保证归一化，也就是不满足概率分布的条件  $0 \leq P(X) \leq 1$ 。要解决这个问题，可以添加归一化因子。

$$P(X) = \frac{1}{Z} (\phi_1 \phi_2 \dots \phi_n) \quad (5.21)$$

其中， $Z = \sum_x (\phi_1 \phi_2 \dots \phi_n)$  称为归一化常数，也称为规范化因子。但在很多时候，往往只考察分子的结果，因此为了区分，用  $\tilde{P}(X)$  来表示非归一化的因子乘积，也就是  $\tilde{P}(X) = \phi_1 \phi_2 \dots \phi_n$ 。

其次，因子分解具有多样性。势函数是衡量局部随机变量相互的影响关系。如图 5.26 所示的无向图模型，利用式(5.21)，读者可能会得到如下的概率分布表达式：

$$P(X) = \frac{\phi_1(X_1, X_3) \phi_2(X_1, X_4) \phi_3(X_2, X_3) \phi_4(X_2, X_5) \phi_5(X_3, X_4) \phi_6(X_3, X_5) \phi_7(X_4, X_5)}{Z} \quad (5.22)$$

在式(5.22)中，把每一条边都作为因子，因此因子分解中含有的因子很多，但仔细观察发现因子的表示存在更合理的表示。考察最大团  $\{X_1, X_3, X_4\}$ ，由团的性质可知，团的各个节点之间都相互关联，因此如果定义一个以此集合为辖域的势函数  $\phi'(X_1, X_3, X_4)$ ，那么  $\phi'$  体现了  $X_1$ 、 $X_3$ 、 $X_4$  三者之间的相互关系，把式(5.22)的因子  $\phi_1(X_1, X_3)$ 、 $\phi_2(X_1, X_4)$  和  $\phi_5(X_3, X_4)$  都统一用  $\phi'(X_1, X_3, X_4)$  来表示。

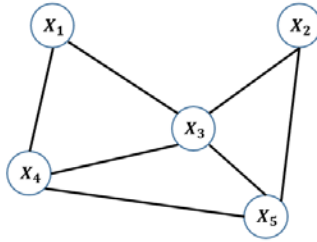


图5.26 无向概率图示例

综上所述，我们得到马尔科夫网络中基于最大团分解的因子分解定理，也称之为 Hammersley-Clifford 定理。

**Hammersley-Clifford 定理：**定义在随机变量集合  $X$  上的马尔科夫网络  $H$ ，其联合概率分布  $P(X)$  可以表示为：

$$P(X) = \frac{1}{Z} \prod_c \phi_c(X_c) \quad (5.23)$$

其中， $C$  是  $H$  中的一个最大团， $X_c$  表示  $C$  中的随机变量集合， $\phi_c(X_c)$  是辖域定义在  $X_c$  上的势函数，也称为**团位势**（Clique Potential）， $Z$  是正则化因子，记为：

$$Z = \sum_X \prod_c \psi_c(X_c) \quad (5.24)$$

### 5.4.2 马尔科夫独立性

与贝叶斯网络一样，马尔科夫网络也蕴含了一系列独立性假设，本节将给出马尔科夫网络的全局独立性定义，为此，首先定义无向图的有效路径。

5.3.4 节给出了有向图的有效迹判断，对于无向图，迹与路径是等价的，故只需要判断有效路径即可，并且相比有效迹判断，有效路径的判断相当简单并且直观。对于马尔科夫网络  $H$ ，随机变量之间的影响是沿着无向边“流动”，但当某些中间节点已知的条件下，这个流动相当于被截断。下面给出判断无向图有效路径的形式化定义。

**定义 5.7：**图  $H$  是定义在随机变量集合  $X = (X_1, X_2, \dots, X_n)$  的一个马尔科夫网络， $X_1 - X_2 - \dots - X_k$  是图  $H$  的一条路径，令  $Z \subseteq X$  为观察集合，如果所有  $X_i (i = 1, 2, \dots, k)$  均不在  $Z$  中，则路径  $X_1 - X_2 - \dots - X_k$  在给定观察集合  $Z$  时是一条有效路径。

基于有效路径定义，可以给出无向图的 D-分离定义。

**定义5.8:** 令 $X$ 、 $Y$ 、 $Z$ 是马尔科夫网络的3个节点集。在给定 $Z$ 的前提下, 假如对任意节点 $x \in X$ 和 $y \in Y$ 之间不存在有效路径, 即 $x - \dots - Z - \dots - y$ 不满足定义5.7的有效性条件, 那么称 $X$ 与 $Y$ 在给定 $Z$ 的前提下是D-分离的, 记为 $\text{d-sep}_H(X; Y|Z)$ ,  $Z$ 也称为分离集。

由D-分离定义的独立性集合记为:

$$I(H) = \{(x \perp y|Z): \text{d-sep}_H(X; Y|Z), x \in X, y \in Y\} \quad (5.25)$$

$I(H)$ 被称为**全局马尔科夫独立性集合**。在图5.27所示的无向图 $H$ 中, 集合 $X = (X_1, X_2, X_3)$ 和集合 $Y = (Y_1, Y_2, Y_3)$ 被 $Z = (Z_1, Z_2)$ 所分离, 故图 $H$ 所蕴含的全局马尔科夫条件独立集合为:

$$I(H) = \left\{ \begin{array}{l} (X_1 \perp Y_1|Z), (X_1 \perp Y_2|Z), (X_1 \perp Y_3|Z), \\ (X_2 \perp Y_1|Z), (X_2 \perp Y_2|Z), (X_2 \perp Y_3|Z), \\ (X_3 \perp Y_1|Z), (X_3 \perp Y_2|Z), (X_3 \perp Y_3|Z) \end{array} \right\}$$

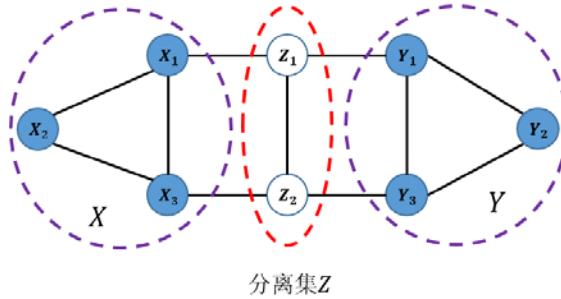


图5.27 由D-分离导出的全局独立性集合

由全局马尔科夫独立性可以推广得到两个局部的独立性质。

- **局部马尔科夫独立性:** 对于任意的马尔科夫网络 $H$ , 它的任意一个节点变量记为 $X$ ,  $X$ 的所有邻接变量集合记为 $\text{adj}(X)$ , 其余变量集记为 $Y$ , 那么 $H$ 蕴含下面的局部马尔科夫独立性约束:

$$I_l(H) = (X \perp y|\text{adj}(X)): \text{d-sep}_H(X; Y|\text{adj}(X)), y \in Y \quad (5.26)$$

如图5.28所示, 对于变量 $X_1$ , 其对应的邻接变量集合为 $\text{adj}(X_1) = \{X_2, X_3, X_4\}$ , 故 $H$ 所蕴含的局部马尔科夫独立性集合为:

$$I_l(H) = \left\{ \begin{array}{l} (X_1 \perp X_5|\{X_2, X_3, X_4\}), (X_1 \perp X_6|\{X_2, X_3, X_4\}), (X_1 \perp X_7|\{X_2, X_3, X_4\}), \\ (X_1 \perp X_8|\{X_2, X_3, X_4\}), (X_1 \perp X_9|\{X_2, X_3, X_4\}) \end{array} \right\}$$

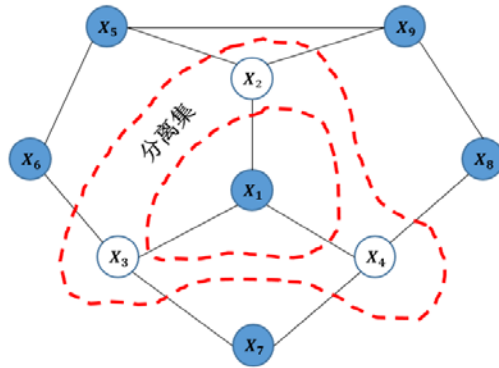


图5.28 局部马尔科夫独立性约束

- **成对马尔科夫独立性:** 对于马尔科夫网络 $H$ , 给定任意两个不相邻的变量 $X$ 和 $Y$ , 其他变量集记为 $Z$ , 那么 $H$ 蕴含成对马尔科夫独立性约束:

$$I_p(H) = \{(X \perp Y | Z)\} \quad (5.27)$$

如图5.29所示, 变量 $h_i$ 和变量 $h_j$ 不相邻, 其余随机变量集合记为 $V = \{v_1, v_2, \dots, v_n\}$ , 故 $H$ 所蕴含的成对马尔科夫独立性集合为:  $(h_i \perp h_j | V)$ 。

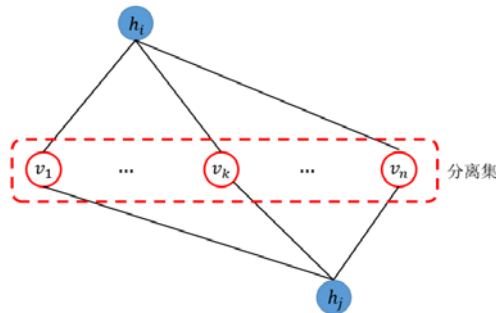


图5.29 成对马尔科夫独立性约束

事实上, 图5.29所示的无向图结构被称为受限玻尔兹曼机, 它是一种特殊的马尔科夫网络模型, 我们将在第11章中重点讲解。

前面已经介绍了两种概率图模型表示, 无独有偶, 在实际的分布中, 利用变量内部的独立性结构, 也就是每一个变量倾向于直接与很少几个其他变量交互的特点, 可以将多变量联合分布用一种很简洁的形式来表示。而推断的目的是在知道某些观察变量值的前提下, 查询未知变量的值, 比如查询在秋季, 一个患者有鼻窦充血和肌肉疼痛症状, 那么他患有流感的概率为:  $p(\text{flu} = \text{true} \mid \text{season} = \text{fall}, \text{congestion} =$

`true, pain = false`)。

推断的策略可以分为精准推断和近似推断。不幸的是，科学家已经证明，精准推断任务在最坏的情况下，其时间复杂度是指数级别的，也就是一个NP-难的问题。更糟糕的是，即使是近似推断也是一个指数级别的复杂度。但幸运的是，在绝大多数情况下，我们遇到的实际问题都不是这种最坏的情况，正如后面将讲解的，对图模型使用精准的或者近似的推理算法，可以非常有效的解决很多实际问题。

首先来形式化给出推断问题的定义：对于图模型 $G$ ，它的随机变量集合由三部分构成，记为 $\{Q, E, W\}$ ，其中 $Q$ 称为查询变量集，记为 $Q = \{Q_1, Q_2, \dots, Q_n\}$ ， $E$ 称为观察变量集，记为 $E = \{E_1 = e_1, E_2 = e_2, \dots, E_m = e_m\}$ ，其余随机变量记为 $W$ 。推断的目标是，在给定观察变量集 $E$ 的前提下，求取查询变量集 $Q$ 的概率 $P(Q|E)$ ，由条件概率公式可得：

$$P(Q|E) = \frac{P(Q, E)}{P(E)} \quad (5.28)$$

其中，对于 $P(Q, E)$ 和 $P(E)$ ，满足：

$$P(Q, E) = \sum_W P(Q, E, W) \quad (5.29)$$

$$P(Q, E) = \sum_E \sum_Q P(Q, E, W) \quad (5.30)$$

推断问题从求取条件概率转换为求取边缘概率分布，因此，求解推断问题的关键是如何高效求取边缘概率分布问题。

## 5.5 变量消除

变量消除（Variable Elimination，简称VE）是最简单、最基础的推断策略。下面通过一个例子来了解变量消除的推导过程。不失一般性，以图5.30的马尔科夫网络来分析变量消除的过程，对于贝叶斯网络以及更复杂的概率图模型，下面的计算过程同样适用。



图5.30 简单的马尔科夫网络

图5.30的网络中, 变量的集合为 $\{A, B, C, D, E\}$ , 其中查询变量为 $\{A\}$ , 观察变量为 $\{E = e\}$ , 计算条件概率 $P(A|E = e)$ 。由式(5.28)得, 求解 $P(A|E = e)$ 问题可以转化为求解边缘概率 $P(A, E)$ 和 $P(E)$ 问题。下面推导边缘概率 $P(E)$ 的计算过程, 由边缘分布的概率公式, 有:

$$P(E) = \sum_A \sum_B \sum_C \sum_D P(A, B, C, D, E) \quad (5.31)$$

无向图的联合概率分布可以由基于最大团的因子分解求得, 图5.30的最大团集合包括:  $Q = \{(A, B), (B, C), (C, D), (D, E)\}$ , 故有:

$$P(A, B, C, D, E) = \frac{\phi_{Q_{A \rightarrow B}}(A, B) \phi_{Q_{B \rightarrow C}}(B, C) \phi_{Q_{C \rightarrow D}}(C, D) \phi_{Q_{D \rightarrow E}}(D, E)}{Z} \quad (5.32)$$

成立。事实上, 并不需要关心 $Z$ 的取值, 因为 $P(A, E)$ 与 $P(E)$ 的分母都为 $Z$ , 利用式(5.28)求解 $P(A|E = e)$ 时, 会消去分母 $Z$ 。故下面我们的重点是计算分子 $\tilde{P}(E)$ 。把式(5.32)代入式(5.31)可得:

$$\begin{aligned} \tilde{P}(E) &= \sum_A \sum_B \sum_C \sum_D \phi_{Q_{A \rightarrow B}}(A, B) \phi_{Q_{B \rightarrow C}}(B, C) \phi_{Q_{C \rightarrow D}}(C, D) \phi_{Q_{D \rightarrow E}}(D, E) \\ &= \sum_B \sum_C \sum_D \phi_{Q_{B \rightarrow C}}(B, C) \phi_{Q_{C \rightarrow D}}(C, D) \phi_{Q_{D \rightarrow E}}(D, E) \sum_A \phi_{Q_{A \rightarrow B}}(A, B) \\ &= \sum_B \sum_C \sum_D \phi_{Q_{B \rightarrow C}}(B, C) \phi_{Q_{C \rightarrow D}}(C, D) \phi_{Q_{D \rightarrow E}}(D, E) \tau_1(B) \end{aligned}$$

记 $\sum_A \phi_{Q_{A \rightarrow B}}(A, B) = \tau_1(B)$ , 可以看到 $\tau_1(B)$ 是一个只含有 $B$ 的函数, 变量 $A$ 通过求和被消除。通过这种方式, 最后得到:

$$\begin{aligned} \tilde{P}(E) &= \sum_B \sum_C \sum_D \phi_{Q_{B \rightarrow C}}(B, C) \phi_{Q_{C \rightarrow D}}(C, D) \phi_{Q_{D \rightarrow E}}(D, E) \tau_1(B) \\ &= \sum_C \sum_D \phi_{Q_{C \rightarrow D}}(C, D) \phi_{Q_{D \rightarrow E}}(D, E) \sum_B \phi_{Q_{B \rightarrow C}}(B, C) \tau_1(B) \\ &= \sum_C \sum_D \phi_{Q_{C \rightarrow D}}(C, D) \phi_{Q_{D \rightarrow E}}(D, E) \tau_2(C) \\ &= \sum_D \phi_{Q_{D \rightarrow E}}(D, E) \sum_C \phi_{Q_{C \rightarrow D}}(C, D) \tau_2(C) \\ &= \sum_D \phi_{Q_{D \rightarrow E}}(D, E) \tau_3(D) \end{aligned}$$

$$= \tau_4(E) \quad (5.33)$$

对于 $P(A, E)$ 采用同样的计算策略，这里不再详解。可以看到，式(5.33)的计算过程，是通过因子交换，把多个因子的积的求和问题转化为局部变量的求积与求和问题，进而不断消除局部变量的过程，因此，把该策略称为**变量消除**策略。

变量消除最大的缺点是计算复杂度很大，假如 $W$ 由100个二值随机变量集合构成，那么求解 $P(Q, E)$ 需要 $2^{100}$ 次求和运算。显然这个时间复杂度在实际的工程应用中是难以接受的。

## 5.6 信念传播

在上一节中，阐述了如何利用图模型的结构来进行变量消除。变量消除的基本思想是利用联合概率分布的因子分解特性，将全局的因子乘积转化为局部的运算，每一步的局部操作都能消除部分随机变量。本节将介绍另一种精准推理算法：信念传播（有些文献也称为置信传播，Belief Propagation），信念传播本质上也是消除变量的过程，但它使用一种更全局的数据结构，从而使得算法比局部变量消除更加高效。

### 5.6.1 聚类图

首先介绍信念传播所使用的数据结构：聚类图（Cluster Graph）。聚类图是一种无向图结构，设随机变量集合 $X = \{X_1, X_2, \dots, X_n\}$ 上的因子集合为 $\Phi = \{\phi_k, k = 1, 2, \dots, n\}$ ，则由其构建的聚类图定义为：

（1）聚类图的第 $i$ 个节点记为 $C_i$ ， $C_i$ 是与变量子集相关联的一个变量聚类，也就是满足 $C_i \subseteq \{X_1, X_2, \dots, X_n\}$ 。具体来说，给图模型的每一个因子 $\phi_k$ 都分配到且只分配到一个簇 $C_i$ 中，故而满足 $\text{scope}(\phi_k) \subseteq \text{Val}(C_i)$ ，然后为每一个节点 $C_i$ 分配一个势函数 $\psi_i(C_i)$ （注意，这里为了与马尔科夫网络的势函数做区分，我们使用了不同的符号来表示）， $\psi_i$ 有时也被称为信念（Belief）。

不失一般性，设被分配到 $C_i$ 的因子集合为 $\{\phi_1, \phi_2, \dots, \phi_k\}$ ，有 $\text{Val}(C_i) = \text{scope}(\phi_1) \cup \text{scope}(\phi_2) \cup \dots \cup \text{scope}(\phi_k)$ ，同时定义 $C_i$ 的势函数 $\psi_i(C_i)$ 的计算公式为：

$$\psi_i(C_i) = \prod_{j=1}^k \phi_j \quad (5.34)$$



(2) 聚类图包含了用来连接两个聚类节点 $C_i$ 和 $C_j$ 的无向边 $S_{i,j}$ , 边 $S_{i,j}$ 与变量子集 $C_i \cap C_j$ 相关联。聚类图中边的作用是用于消息传播。聚类图为每一条边 $S_{i,j}$ 分配两个消息函数 $\delta_{i \rightarrow j}(S_{i,j})$ 和 $\delta_{j \rightarrow i}(S_{j,i})$ , 其中 $\delta_{i \rightarrow j}(S_{i,j})$ 表示从 $C_i$ 向 $C_j$ 发送的消息, 也就是消息传递, 初始化为 $\delta_{i \rightarrow j}(S_{i,j}) = 1$ 。 $\delta_{i \rightarrow j}(S_{i,j})$ 的计算过程: 设 $C_i$ 的势函数为 $\psi_i(C_i)$ ,  $C_i$ 的所有相邻节点集合记为 $N_i$ , 从 $C_i$ 向 $C_j$ 发送的消息 $\delta_{i \rightarrow j}(S_{i,j})$ 满足:

$$\delta_{i \rightarrow j}(S_{i,j}) = \sum_{C_i - S_{i,j}} \left( \psi_i \times \prod_{k \in \{N_i - C_j\}} \delta_{k \rightarrow i} \right) \quad (5.35)$$

把式(5.35)的计算过程用图5.31来可视化表示。

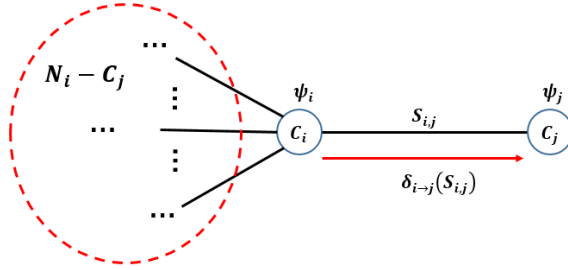


图5.31 消息函数计算

通过一个例子来考察聚类图的性质。假设由随机变量集合 $\{X_1, X_2, \dots, X_n\}$ 构成的概率图模型中, 包括的因子 $\phi$ 为:

$$\phi = \{\phi_1(A, B, C), \phi_2(B, C), \phi_3(B, D), \phi_4(D, E), \phi_5(B, E), \phi_6(B, D, F)\} \quad (5.36)$$

那么由该因子集合构成的一个聚类图, 如图5.32所示。

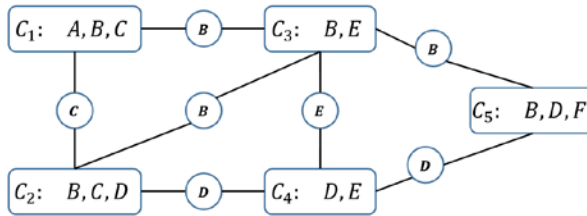


图5.32 满足式5.27因子集合的聚类图

其中,  $\psi_1(C_1) = \phi_1(A, B, C)$ ,  $\psi_2(C_2) = \phi_2(B, C)\phi_3(B, D)$ ,  $\psi_3(C_3) = \phi_5(B, E)$ ,  $\psi_4(C_4) = \phi_4(D, E)$ ,  $\psi_5(C_5) = \phi_6(B, D, F)$ 。

但仔细观察, 我们发现因子的分配可以有多种选择, 比如 $\phi_2(B, C)$ 既可以添加到

簇 $C_1$ ，也可以添加到簇 $C_2$ 中。也就是说对于同一个聚类图，每一个节点的势函数表示不唯一。下面是图5.32的另一个节点势函数表示：

$$\begin{aligned}\psi_1(C_1) &= \phi_1(A, B, C)\phi_2(B, C), \quad \psi_2(C_2) = \phi_3(B, D), \quad \psi_3(C_3) = \phi_5(B, E), \\ \psi_4(C_4) &= \phi_4(D, E), \quad \psi_5(C_2) = \phi_6(B, D, F)\end{aligned}$$

此外，除了节点势函数的定义具有多样性之外，边 $S_{i,j}$ 所蕴含的传递变量集合也可以有多种形式，图5.33是满足式(5.36)因子分解的另一个聚类图。图5.33与图5.32相比，发生了两处改变：对于边 $S_{1,2}$ ，它蕴含的变量集合从原来的 $\{C\}$ 变为 $\{B, C\}$ ，也就是从 $C_1$ 向 $C_2$ 传递消息，传播的变量集从 $\{C\}$ 变为 $\{B, C\}$ ；边 $S_{2,3}$ 被删除，也就是 $C_3$ 与 $C_2$ 之间不会发生消息传递。

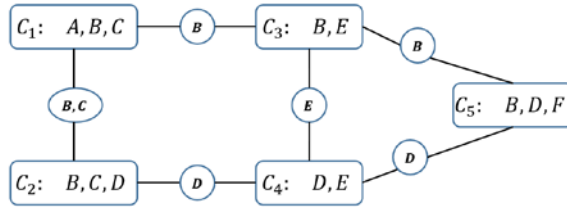


图5.33 满足式(5.27)因子集合的另一个聚类图

既然满足同一个因子集合的聚类图可以有多种不同的构造方式，那么应该怎么样判断一个聚类图是否合法呢？下面给出判断一个合法的聚类图应该具有的两个重要性质。

(1) 给定概率图模型的因子集合 $\Phi = \{\phi_1, \phi_2, \dots, \phi_k, \dots, \phi_n\}$ ，每一个因子 $\phi_k$ 都被分配到且只分配到一个簇 $C_k$ 中，满足 $\text{scope}(\phi_k) \subseteq \text{Val}(C_k)$ 。

(2) 对聚类图中任意两个的节点 $C_i$ 和 $C_j$ ，若随机变量 $X \in C_i \cap C_j$ ，那么在 $C_i$ 与 $C_j$ 之间存在且仅存在一条路径： $C_i - \dots - C_a - S_{a,b} - C_b - \dots - C_j$ ，使得该路径上的任意一个节点 $C_a$ 和边 $S_{a,b}$ ，均满足 $X \in \text{Val}(C_a)$ ， $X \in \text{Val}(S_{a,b})$ 。

图5.34和图5.35分别展示了两个非法的聚类图，其中图5.34是非法的，因为该聚类图不满足性质(2)中路径存在的性质，从 $C_5$ 到 $C_2$ 不存在一条包含变量 $B$ 的路径；图5.35是非法的，因为该聚类图不满足性质(2)中路径唯一的性质，对于变量 $B$ ，存在两条从 $C_5$ 到 $C_2$ 的路径，使得路径上的节点和边都包含变量 $B$ 。这两条路径分别为 $\text{path}_1 = C_5 - S_{5,3} - C_3 - S_{3,1} - C_1 - S_{1,2} - C_2$ 和 $\text{path}_2 = C_5 - S_{5,3} - C_3 - S_{3,2} - C_2$ 。

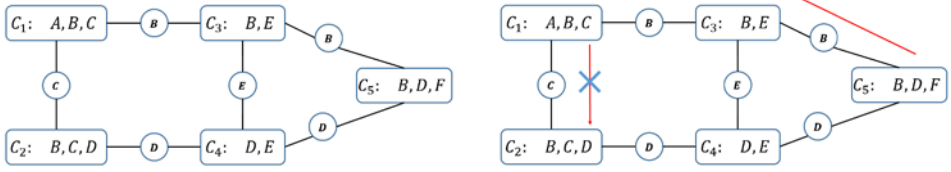


图5.34 非法的聚类图，这是因为 $C_5 = \{B, D, F\}$ ， $C_2 = \{B, C, D\}$ ，故 $\{B\} \subseteq C_2 \cap C_5$ ，但从 $C_2$ 到 $C_5$ 不存在一条路径 $C_2 - \dots - C_i - S_{i,j} - C_j - \dots - C_5$ ，使得 $C_i$ ， $C_j$ 和 $S_{i,j}$ 均包含变量 $B$

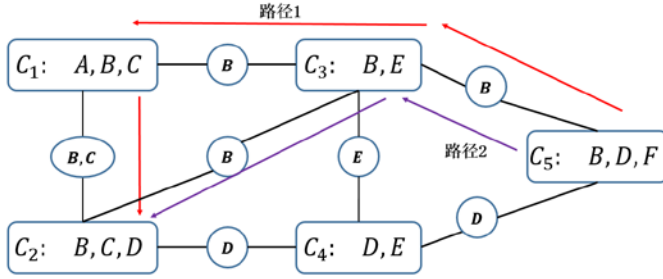


图5.35 非法的聚类图，这是因为 $C_5 = \{B, D, F\}$ ， $C_2 = \{B, C, D\}$ ，故 $\{B\} \subseteq C_2 \cap C_5$ ，但从 $C_2$ 到 $C_5$ 之间存在两条路径 $\text{path}_1 = C_5 - S_{5,3} - C_3 - S_{3,1} - C_1 - S_{1,2} - C_2$ ， $\text{path}_2 = C_5 - S_{5,3} - C_3 - S_{3,2} - C_2$ ，使得路径上的任意 $C_i$ 和 $S_{i,j}$ 均包含变量 $B$

下面给出在聚类图上执行信念传播算法的步骤，如图5.36所示，算法最后得到的 $B_i(C_i)$ 就是变量集合 $\text{Val}(C_i)$ 的非归一化联合概率分布 $\tilde{P}(\text{Val}(C_i))$ 。

### 算法5.3 信念传播算法

1. 由无向图马尔科夫网构建聚类图，把马尔科夫网中的因子分配到聚类图的某一个节点 $C_k$ 中
2. 初始化聚类图的每一个节点的初始势 $\psi_i(C_i) = \prod_k \phi_k$ ，也就是在构建聚类图时，所有被分配到 $C_k$ 的无向图因子
3. 初始化所有的消息 $\delta_{i \rightarrow j} = 1$
4. 重复执行下面的操作，直至收敛

选择边 $S_{i,j}$ ，计算其消息传递：

$$\delta_{i \rightarrow j}(S_{i,j}) = \sum_{C_i - S_{i,j}} \psi_i \times \prod_{k \in \{N_i - C_j\}} \delta_{k \rightarrow i}$$

5. 对于聚类图的每一个节点 $C_i$ ，得到 $C_i$ 对应变量的联合概率分布(非规范化)为：

$$B_i(C_i) = \tilde{P}(\text{Val}(C_i)) = \psi_i \times \prod_{k \in \{N_i\}} \delta_{k \rightarrow i}$$

图5.36 信念传播算法

### 5.6.2 团树

5.6.1节介绍了信念传播算法所使用的聚类树数据结构，本节把聚类树应用到贝叶斯网络或马尔科夫网络中。在贝叶斯网络中，联合概率分布是基于条件概率的乘积：

$$P(X_1, X_2, \dots, X_n) = \prod_{i=1}^n P(X_i | X_1, X_2, \dots, X_{i-1})$$

在马尔科夫网络中，联合概率分布是基于最大团的势函数的乘积：

$$P(X_1, X_2, \dots, X_n) = \frac{1}{Z} \prod_c \phi_c(X_c), \quad Z = \sum_x \prod_c \phi_c(X_c)$$

由这种团因子构建聚类图模型，称为团树（Clique Tree），也就是说在贝叶斯网络或马尔科夫网络中构建聚类图时，得到的是一棵树。在团树上执行信念传播算法也被称为团树算法。

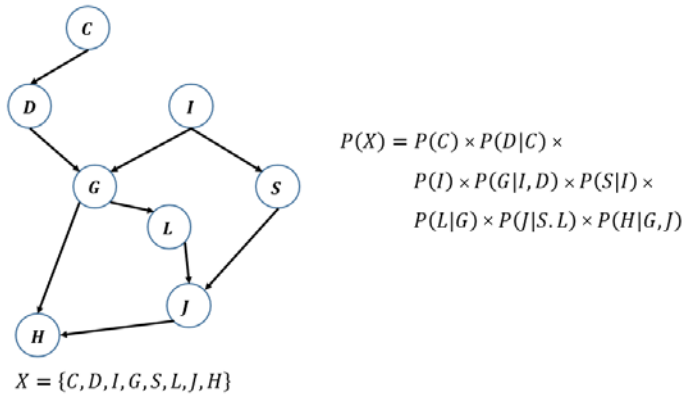


图5.37 贝叶斯网络G及其联合概率分布

图5.37是一个贝叶斯网络及其联合概率分布的表示，构建对应的团树结构如图5.38所示，边的消息传递定义如图5.39所示，节点的势函数定义如下：

$$\psi_1(C_1) = P(C) \times P(D|C) \quad (5.37)$$

$$\psi_2(C_2) = P(G|I, D) \quad (5.38)$$

$$\psi_3(C_3) = P(I) \times P(S|I) \quad (5.39)$$

$$\psi_4(C_4) = P(J|L, S) \times P(L|G) \quad (5.40)$$

$$\psi_5(C_5) = P(H|J, G) \quad (5.41)$$

通过一个查询来验证团树算法的正确性, 假设现在要计算边缘分布 $P(G, I, D)$ , 也就是需要证明 $P(G, I, D) = B_2(C_2) = \psi_2(C_2) \times \delta_{1 \rightarrow 2}(D) \times \delta_{3 \rightarrow 2}(G, I)$ 成立。化简:

$$\begin{aligned}
 \psi_2(C_2) \times \delta_{1 \rightarrow 2}(D) \times \delta_{3 \rightarrow 2}(G, I) &= \psi_2(C_2) \times \sum_C \psi_1(C_1) \times \sum_S (\psi_3(C_3) \times \delta_{4 \rightarrow 3}(G, S)) \\
 &= \psi_2(C_2) \times \sum_C \psi_1(C_1) \times \sum_S \left( \psi_3(C_3) \times \left( \sum_{J, L} (\psi_4(C_4) \times \delta_{5 \rightarrow 4}(G, J)) \right) \right) \\
 &= \psi_2(C_2) \times \sum_C \psi_1(C_1) \times \sum_S \left( \psi_3(C_3) \times \left( \sum_{J, L} \left( \psi_4(C_4) \times \sum_H \psi_5(C_5) \right) \right) \right) \quad (5.42)
 \end{aligned}$$

把式(5.37)至式(5.41)分别代入式(5.42):

$$\begin{aligned}
 &\psi_2(C_2) \times \delta_{1 \rightarrow 2}(D) \times \delta_{3 \rightarrow 2}(G, I) \\
 &= P(G|I, D) \times \left( \sum_C P(C) \times P(D|C) \right) \\
 &\quad \times \left( \sum_S \left( P(I) \times P(S|I) \right. \right. \\
 &\quad \times \left. \left. \left( \sum_{J, L} \left( P(J|L, S) \times P(L|G) \times \sum_H P(H|J, G) \right) \right) \right) \right) \\
 &= P(G|I, D) \times P(D) \times \left( \sum_S \left( P(I) \times P(S|I) \times \left( \sum_{J, L} (P(J|L, S) \times P(L|G)) \right) \right) \right) \\
 &= P(G|I, D) \times P(D) \times \left( \sum_S \left( P(I) \times P(S|I) \times \left( \sum_L P(L|G) \times \sum_J P(J|L, S) \right) \right) \right) \\
 &= P(G|I, D) \times P(D) \times P(I) = P(G, I, D)
 \end{aligned}$$

证毕。其中最后一步是由随机变量 $I$ 与随机变量 $D$ 相互独立的性质得到, 同样可以对任意的变量子集进行验证, 这里不再详述。

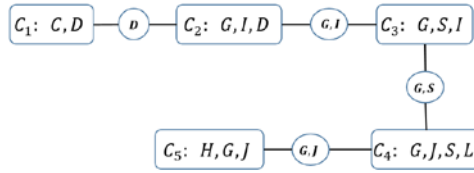


图5.38 由图5.36生成的团树

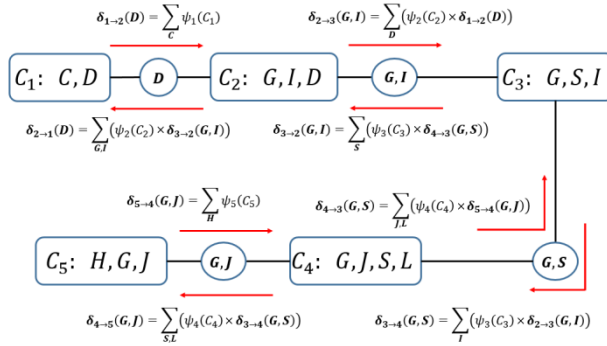


图5.39 团树的消息传递

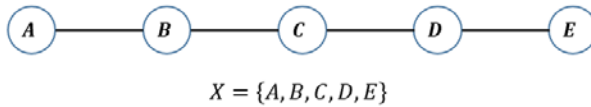
通过马尔科夫网络的例子来考察无向图模型与团树的关系，考察图5.40的马尔科夫网络 $H$ ，由其创建的一个团树如图5.41所示，边的消息传递定义如图5.42所示，定义节点的势函数定义如下：

$$\psi_1(C_1) = \phi_1(A, B) \quad (5.43)$$

$$\psi_2(C_2) = \phi_2(B, C) \quad (5.44)$$

$$\psi_3(C_3) = \phi_3(C, D) \quad (5.45)$$

$$\psi_4(C_4) = \phi_4(D, E) \quad (5.46)$$



$$P(X) = \phi_1(A, B) \times \phi_2(B, C) \times \phi_3(C, D) \times \phi_4(D, E)$$

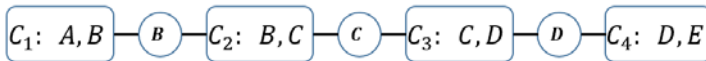
图5.40 马尔科夫网络 $H$ 及其联合概率分布

图5.41 由图5.39生成的团树

下面我们同样通过一个查询来验证团树算法在无向图模型上的正确性。假设要计算边缘分布 $P(C, D)$ ，由于 $P$ 满足 $P(C, D) \propto \tilde{P}(C, D)$ ，我们不考虑归一化因子，也就是需要验证 $\tilde{P}(C, D) = B_3(C_3) = \psi_3(C_3) \times \delta_{2 \rightarrow 3}(C) \times \delta_{4 \rightarrow 3}(D)$ ，成立，化简：

$$\begin{aligned}
 & \psi_3(C_3) \times \delta_{2 \rightarrow 3}(C) \times \delta_{4 \rightarrow 3}(D) \\
 &= \phi_3(C, D) \times \left( \sum_B (\psi_2(C_2) \times \delta_{1 \rightarrow 2}(B)) \right) \times \left( \sum_E \psi_4(C_4) \right) \\
 &= \phi_3(C, D) \times \left( \sum_B \left( \phi_2(B, C) \times \sum_A \phi_1(A, B) \right) \right) \times \left( \sum_E \phi_4(D, E) \right) \quad (5.47)
 \end{aligned}$$

式(5.47)化简后，是一个只含有变量 $C$ 与变量 $D$ 的函数，与变量消除的结果一致，故 $\tilde{P}(C, D) = B_3(C_3)$ 成立。

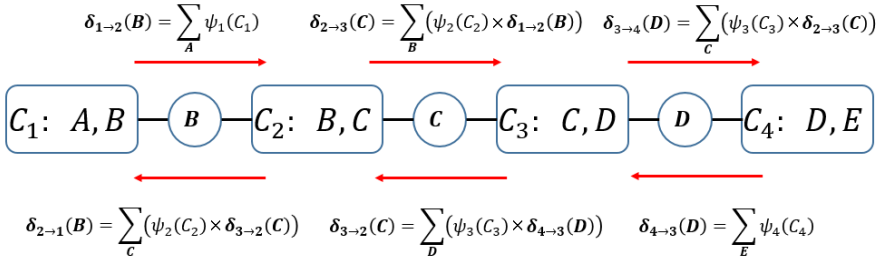


图5.42 对应于图5.40的团树的消息传递

### 5.6.3 由变量消除构建团树

我们已经介绍了团树算法的过程和它的正确性，但到目前为止，仍然有一个基本的问题没有解决，由5.6.1节可知，合法的团树（聚类图）的构建有很多种方式，那么应该如何构建团树呢？事实上，团树的构建并不是唯一的，本节介绍一种通用的从变量消除的顺序来构建团树的方法。变量消除与团树之间的关系如图5.43所示。

	变量消除	团 树
$\psi_i$	由贝叶斯网络或马尔科夫网络中，多个因子 $\phi$ 组合而成的新的组合因子	团树中节点 $C_i$ 的势函数
$\tau_i$	对 $\psi_i$ 进行变量累加后消除部分变量，形成的新的因子	消息，由团 $C_i$ 产生的消息，用于向其它的团 $C_j$ 传播消息

图5.43 变量消除与团树之间的关系

从图5.43可以观察到变量消除与团树算法之间的关系。团树的每一个节点 $C_i$ 的势函数可以由两部分构成，包括贝叶斯网络或马尔科夫网络中的因子 $\phi_i$ ，以及消息 $\tau_i$  ( $\tau_i$ 是5.5节中使用的记号，用于表示变量消除后产生的新的因子，这个新的因子对应于团树中的消息 $\delta_{i \rightarrow j}$ )。当 $C_j$ 的势函数的计算需要使用消息 $\tau_i$ 时，那么需要在 $C_i$ 与 $C_j$ 之间构建一条边 $S_{i,j}$ ， $S_{i,j}$ 从 $C_i$ 指向 $C_j$ 方向传递的消息为 $\delta_{i \rightarrow j}(S_{i,j}) = \tau_i$ 。

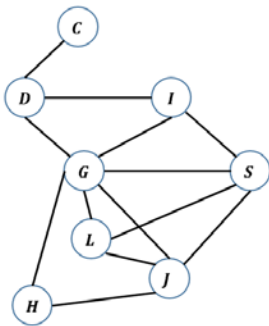


图5.44 马尔科夫网络示例，图中的节点集合为 $X = \{C, D, G, I, S, L, J, H\}$ ，最大团集合包括  $\{(C, D), (G, D, I), (G, S, I), (G, S, J, L), (G, H, J)\}$

以图5.44为例来考察如何根据变量消除的顺序来构建一棵团树。图5.44的马尔科夫网络对应的联合概率分布为：

$$P(C, D, G, I, S, L, J, H) = \frac{\phi_1(C, D)\phi_2(G, I, D)\phi_3(G, I, S)\phi_4(G, J, S, L)\phi_5(G, H, J)}{Z}$$

现在查询 $P(J)$ ，对应的VE算法中，变量消除的顺序如下。

- (1) 消除变量 $C$ ：变量 $C$ 只包含在因子 $\phi_1(C, D)$ 中，故有：



$$\begin{aligned}\psi_1(C, D) &= \phi_1(C, D) \\ \tau_1(D) &= \sum_C \psi_1(C, D)\end{aligned}$$

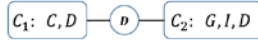
由 $\psi_1$ 首先确定簇 $C_1$ ,  $\text{Val}(C_1) = \{C, D\}$ :

$$C_1: C, D$$

(2) 消除变量 $D$ : 变量 $D$ 包含在因子 $\phi_2(G, I, D)$ , 以及新生成的 $\tau_1(D)$ 中, 故有:

$$\begin{aligned}\psi_2(G, I, D) &= \phi_2(G, I, D)\tau_1(D) \\ \tau_2(G, I) &= \sum_D \psi_2(G, I, D)\end{aligned}$$

由 $\psi_2$ 确定簇 $C_2$ ,  $\text{Val}(C_2) = \{G, I, D\}$ , 且 $\psi_2$ 含有由 $C_1$ 传递过来的消息 $\tau_1(D)$ , 故 $C_1$ 与 $C_2$ 之间有边相连:



(3) 消除变量 $I$ : 变量 $I$ 包含在因子 $\phi_3(G, I, S)$ , 以及新生成的 $\tau_2(G, I)$ 中, 故有:

$$\begin{aligned}\psi_3(G, I, S) &= \phi_3(G, I, S)\tau_2(G, I) \\ \tau_3(G, S) &= \sum_I \psi_3(G, I, S)\end{aligned}$$

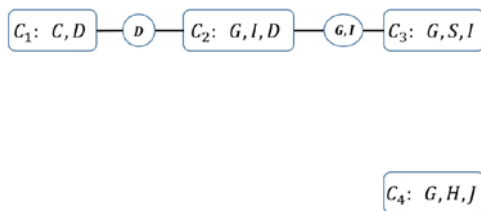
由 $\psi_3$ 确定簇 $C_3$ ,  $\text{Val}(C_3) = \{G, I, S\}$ , 且 $\psi_3$ 含有由 $C_2$ 传递过来的消息 $\tau_2(G, I)$ , 故 $C_2$ 与 $C_3$ 之间有边相连:



(4) 消除变量 $H$ , 变量 $H$ 包含在因子 $\phi_5(G, H, J)$ 中, 它没有与任何当前的其他节点有相交节点, 故有:

$$\begin{aligned}\psi_4(G, H, J) &= \phi_5(G, H, J) \\ \tau_4(G, J) &= \sum_H \psi_4(G, H, J)\end{aligned}$$

由 $\psi_4$ 确定簇 $C_4$ ,  $\text{Val}(C_4) = \{G, H, J\}$ , 但 $\psi_4$ 没有从其他节点传递过来的消息, 故簇 $C_4$ 当前是一个孤立点:



(5) 消除变量  $G$ 、 $S$ 、 $L$ ， $G$ 、 $S$ 、 $L$  包含在由余因子  $\phi_4(G, J, S, L)$ ，以及新生成的  $\tau_3(G, S)$  和  $\tau_4(G, J)$  中，可以统一执行消除操作：

$$\psi_5(G, J, S, L) = \phi_4(G, J, S, L) \times \tau_3(G, S) \times \tau_4(G, J)$$

$$\tau_5(J) = \sum_{G, S, L} \psi_5(G, J, S, L)$$

由  $\psi_5$  确定簇  $C_5$ ， $Val(C_5) = \{G, J, S, L\}$ ，且  $\psi_5$  含有由  $C_3$  传递过来的消息  $\tau_3(G, S)$ ，以及由  $C_4$  传递过来的消息  $\tau_4(G, J)$ ，故  $C_5$  与  $C_3$ 、 $C_4$  之间均有边相连，这就确定了最终的团树结构，如图 5.45 所示。

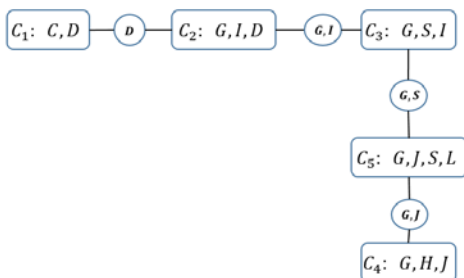


图 5.45 最终的团树结构

## 5.7 MCMC 采样原理

前两节分别介绍了两种精准推断算法，精准推断算法可以被有效地执行，但随着随机变量数量的增长，时间复杂度也呈现指数级增长的趋势，因此在高维空间中是难以被处理的。近似推断试图找到一种时间复杂度比较低，但是又能给出近似正确的解的方法。本节将介绍一种应用非常广泛的近似推断算法：马尔科夫链蒙特卡洛采样算法（Markov Chain Monte Carlo, MCMC）。

### 5.7.1 随机采样

随机采样也被称为蒙特卡洛采样 (Monte Carlo Sampling), 是20世纪40年代发展起来的一门新学科。它最早起源于与原子弹制造相关的曼哈顿计划, 当时包括冯·诺依曼在内的几个计算机科学家, 在美国洛斯阿拉莫斯国家实验室研究裂变物质的中子连锁反应的时候, 开始使用统计模拟的方法, 并在最早的计算机上进行编程实现。

随机采样的一个重要问题就是, 给定一个概率分布 $p(x)$ 如何能够在计算机中生成它的样本。对于简单的概率分布, 例如后面我们将看到的均匀分布和高斯分布, 有一些特殊的方法来高效解决, 但很多时候 $p(x)$ 的形式是很复杂的, 需要一些更有技巧的处理方法。

#### 1. 线性同余伪随机数

一切随机采样方法都以伪随机数为基础, 但由于计算机本身并不会产生随机数, 因此需要一种策略来让计算机高效生成伪随机数。

线性同余是当前计算机中普遍采用的伪随机数生成算法, 由D.H. Lehmer在1949年提出<sup>[5,6]</sup>, 线性同余的计算公式非常简单, 对于任意给定的参数 $a, c, m$ 和初始值 $X_0$ , 线性同余的迭代公式满足:

$$X_{n+1} = (aX_n + c) \bmod m, \quad n \geq 0 \quad (5.48)$$

每一次调用上面的公式, 都会产生一个数值, 并由此数值产生下一个数, 反复循环, 最终会得到一个无穷序列 $\{X_0, X_1, \dots, X_n, \dots\}$ , 容易知道它是一个周期序列, 周期的最大值为 $m$ 。一个优秀的伪随机数产生器, 应该使得周期越大越好。

如何能够构造一个合理的线性同余发生器呢? 从上式, 我们不难发现构造伪随机数产生器的关键是要选择合理的参数 $a, c, m$ , 以及初始值 $X_0$ , 文献[6]给出了下面的参数选择方案, 并证明了该方案可以使得式(5.48)产生的序列周期最大。

**定理:** 由参数 $a, c, m$ 和初始值 $X_0$ 定义的伪随机序列迭代公式(5.48), 它具有最大的周期 $m$ , 当且仅当它满足下面3个条件时。

- 参数 $c$ 与参数 $m$ 互余。
- 对于 $m$ 的任意一个素因子 $p$ , 均满足 $p$ 整除 $(a - 1)$ 。
- 若 $m$ 是4的倍数, 那么 $a$ 也是4的倍数。

如果读者想了解该定理的详细证明过程, 可以参考Knuth的《计算机程序设计艺

术》第二卷第三章，这里不再详述。

## 2. 均匀分布和高斯分布

要产生服从均匀分布Uniform[0,1]的样本,可以由式(5.48)直接生成一个伪随机数 $x$ , 满足 $x = (ax_n + c) \bmod m$ , 值域为 $[0, m - 1]$ , 然后除以 $(m - 1)$ , 得到的值 $u = x/(m - 1)$ , 数值 $u$ 就可以看成是在区间 $[0, 1]$ 上服从均匀分布的采样, 即 $u \sim \text{Uniform}[0,1]$ 。

由均匀分布, 得到高斯分布的高效采样算法。

**Box-Muller变换:** 如果随机变量 $U_1$ 、 $U_2$ 独立, 且 $U_1$ 、 $U_2$ 均服从区间 $[0, 1]$ 上的均匀分布, 定义 $Z_1$ 与 $Z_2$ 的计算公式为:

$$Z_1 = \cos(2\pi U_2) \sqrt{-2 \ln U_1} \quad (5.49)$$

$$Z_2 = \sin(2\pi U_2) \sqrt{-2 \ln U_1} \quad (5.50)$$

那么 $Z_1$ 和 $Z_2$ 相互独立, 且服从标准正态分布。

均匀分布与高斯分布是统计学中最基础的两种分布, 很多复杂的分布通常是以它们为基础来构建。

## 5.7.2 随机过程与马尔科夫链

从19世纪开始, 概率论的研究已经从单一的随机变量 $x$ 延伸到对随机变量的时间序列 $x_1, x_2, x_3, \dots, x_t, \dots$ , 即**随机过程**的研究。设 $\mathbf{T}$ 是一无限整数集, 对于任意的 $t \in \mathbf{T}$ ,  $x_t$ 均为随机变量,  $x_t$ 所有可能取值的集合称为随机过程的**状态空间**。

随机过程的研究要比随机变量复杂很多, 因为对于任意一个时刻的状态 $x_t$ , 它的取值可能与其他各个状态都有关。俄罗斯著名的数学家马尔科夫为了简化问题, 引入了马尔科夫性 (或称为无后效性), 即当前的状态 $x_t$ 仅与它前面的一个状态 $x_{t-1}$ 有关, 而与 $x_{t-1}$ 前面的状态无关, 即有:

$$p(x_{t+1} = x | x_t, x_{t-1}, \dots, x_1) = p(x_{t+1} = x | x_t) \quad (5.51)$$

满足式(5.51)假设的随机过程称为**马尔科夫过程**, 当时间 $t$ 和状态空间都离散时, 也把马尔科夫过程称为**马尔科夫链** (Markov Chain), 简称**马氏链**。

从时刻 $t$ 到时刻 $(t + 1)$ 的状态之间的转移, 用一个矩阵 $\mathbf{p}$ 来表示,  $\mathbf{p}$ 也被称为**状态**

转移矩阵，满足：

$$p(x_{t+1} = j | x_t = i) = p_{ij} \quad (5.52)$$

设随机过程的状态空间大小为 $n$ ，那么显然转移矩阵 $\mathbf{p}$ 是一个大小为 $n \times n$ 的矩阵。转移矩阵 $\mathbf{p}$ 满足下面的两个性质。

- $0 \leq p_{ij} \leq 1$ 。
- $\sum_j p_{ij} = 1$ ，即矩阵中每一行转移概率之和等于1。

假设状态空间有 $n$ 个可能的取值，给定初始概率分布 $\pi_0$ ，且满足 $\pi_0 = (\pi_0(1), \pi_0(2), \dots, \pi_0(n))$ ，其中 $\pi_0(i)$ 表示在初始时刻状态取值为 $i$ 的概率，那么当给定初始概率分布 $\pi_0$ 和状态转移矩阵 $\mathbf{p}$ ，可以得到下一时刻的状态取值分布 $\pi_1 = \pi_0 \mathbf{p}$ ，依此类推，可以得到第 $n$ 时刻的概率分布计算公式为：

$$\pi_n = \pi_0 \mathbf{p}^n \quad (5.53)$$

下面首先来考察马氏链的两个具体例子，然后由这两个例子得到一个一般性结论，最后由这个结论引出非常重要的马氏链定理。

- 例一。

从经济收入的角度，社会学家把人类社会划分为：下层、中层和上层三大人群，我们分别用数字1、2、3来代表这3个阶层。社会学家们还发现决定一个人处于何种收入阶层的最重要的因素是其父母所处的收入阶层。定义状态转移矩阵 $\mathbf{p}$ 如下所示，不难发现，如果一个人的收入属于下层类别，那么他的孩子也属于下层收入阶层的概率比较大，达到0.65，而属于上层收入阶层的概率仅为0.07。状态转移图，如图5.46所示。

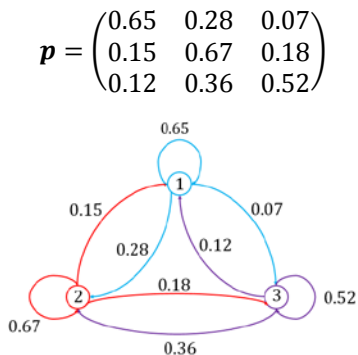


图5.46 状态转移图

假设初始的概率分布为 $\pi_0 = (0.21, 0.68, 0.11)$ ，利用式(5.53)的递推公式，可以得到每一时刻的状态概率分布，如图5.47所示。

第n次迭代	下层	中层	上层
0	0.21	0.68	0.11
1	0.252	0.554	0.194
2	0.270	0.512	0.218
3	0.278	0.497	0.225
4	0.282	0.490	0.226
5	0.285	0.489	0.225
6	0.286	0.489	0.225
7	0.286	0.489	0.225
8	0.286	0.489	0.225
...	...	...	...

图5.47 由递推公式5.53，得到每一时刻的状态概率分布

仔细观察图5.47，我们发现从第五代人开始，状态概率分布就稳定在一个固定值，这会是一种偶然现象吗？带着这个疑问，继续考察另外一个马氏链例子。

• 例二。

考察状态空间包含5个整数值 $\{-2, -1, 0, 1, 2\}$ 的一个马尔科夫链，这些整数以点的形式排成一行。现假设有一个蚱蜢从某一点出发，其状态转移存在以下的规律：在每一个时刻 $x_t$ ，它以0.5的概率留在原地，分别以0.25的概率向左或者向右移动，即满足：

$$p(x_{t+1} = x | x_t = x) = 0.5$$

$$p(x_{t+1} = x - 1 | x_t = x) = p(x_{t+1} = x + 1 | x_t = x) = 0.25$$

但是由于在两端的位置被阻挡，所以在首尾两端蚱蜢的移动方向只有两个，其状态转移如下所示：

$$p(x_{t+1} = 2 | x_t = 2) = p(x_{t+1} = -2 | x_t = -2) = 0.75$$

$$p(x_{t+1} = 1 | x_t = 2) = p(x_{t+1} = -1 | x_t = -2) = 0.25$$

将上面的表述用状态转移矩阵 $\mathbf{p}$ 来表示，则对应的状态转移图如图5.48所示。

$$\mathbf{p} = \begin{pmatrix} 0.75 & 0.25 & 0.0 & 0.0 & 0.0 \\ 0.25 & 0.5 & 0.25 & 0.0 & 0.0 \\ 0.0 & 0.25 & 0.5 & 0.25 & 0.0 \\ 0.0 & 0.0 & 0.25 & 0.5 & 0.25 \\ 0.0 & 0.0 & 0.0 & 0.25 & 0.75 \end{pmatrix}$$

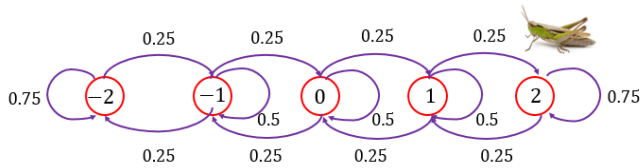


图5.48 状态转移图

假设初始状态概率分布为 $\pi_0 = (0.0, 0.0, 0.0, 0.0, 1.0)$ ，则蚱蜢的初始状态位于点2，利用式(5.53)的递推公式求得每一时刻的状态概率分布如图5.49所示。

第n次迭代	-2	-1	0	1	2
0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.0	0.0	0.25	0.75
2	0.0	0.0	0.0625	0.3125	0.0625
...	...	...	...	...	...
50	0.1976	0.19852	0.2	0.2015	0.2024
...	...	...	...	...	...
200	0.2	0.2	0.2	0.2	0.2
201	0.2	0.2	0.2	0.2	0.2
...	...	...	...	...	...

图5.49 由递推公式(5.53)，得到蚱蜢在每一时刻的状态概率分布

观察图5.49，我们发现，尽管与上一个例子相比，它收敛所需要的迭代次数要更多，但不变的是，概率分布最终都稳定在一个固定值上。

通过上面两个例子，发现一个规律：对于任意给定的转移概率矩阵和初始的概率分布，都存在这种稳定分布，这个收敛性构成了马氏链定理。

**马氏链定理：**对于一个非周期的马氏链，设其转移概率矩阵为 $\mathbf{p}$ ，且任意两个状态之间是相互连通的，那么 $\lim_{n \rightarrow \infty} p_{ij}^n$ 存在且与 $i$ 无关，记 $\lim_{n \rightarrow \infty} p_{ij}^n = \pi(j)$ ，有：

$$\lim_{n \rightarrow \infty} p^n = \begin{pmatrix} \pi(1) & \pi(2) & \dots & \pi(j) & \dots \\ \pi(1) & \pi(2) & \dots & \pi(j) & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \pi(1) & \pi(2) & \dots & \pi(j) & \dots \\ \dots & \dots & \dots & \dots & \dots \end{pmatrix}$$

$$\pi(j) = \sum_{i=0}^{\infty} \pi(i) p_{ij}$$

也就是说  $\pi = (\pi(1), \pi(2), \dots, \pi(j), \dots)$  是方程  $\pi = \pi p$  的唯一解，且等式：

$$\sum_{i=1}^{\infty} \pi(i) = 1$$

成立，我们称  $\pi$  为该马氏链的**平稳分布**。

### 5.7.3 MCMC采样

回到随机采样的问题，假设目标是要生成  $p(x)$  的样本。由于马氏链存在平稳分布的特性，1953年Metropolis提出了一个大胆设想：如果能构造出一个转移矩阵为  $Q$  的马氏链，使得该马氏链的平稳分布  $\pi$  恰好是  $p(x)$ ，那么我们可以从任何一个初始状态  $\pi_0$  出发，沿着马氏链的转移概率矩阵  $Q$  进行转移，得到一个转移序列  $\{\pi_0, \pi_1, \dots, \pi_n, \dots\}$ ，如果在第  $n$  步达到平稳分布，那么  $\pi_n$  及此后的状态  $\pi_{n+1}, \pi_{n+2}, \dots$ ，均可以看成是服从概率分布  $p(x)$  的采样结果。这种方法被称为马尔科夫链蒙特卡洛采样（Markov Chain Monte Carlo Sampling），简称**MCMC采样**。

马氏链的平稳分布  $\pi$  主要由状态转移矩阵  $Q$  决定，通常情况下，在遇到的实际问题中，对应的状态空间都非常大，不可能像上面两个例子一样，直接写出矩阵  $Q$  的具体表达形式，所以MCMC采样的关键问题是如何构造转移矩阵  $Q$ ，使得平稳分布恰好是  $p(x)$ 。首先引入非常重要的细致平稳条件概念。

**细致平稳条件：**设有转移矩阵  $Q$  和分布  $\pi$ ，其中  $Q_{ij} = q(x_{t+1} = j | x_t = i)$  表示从状态  $i$  转移到状态  $j$  的概率， $\pi = (\pi(1), \pi(2), \dots, \pi(i), \dots)$ ， $\pi(i)$  表示取值为状态  $i$  的概率。如果对于所有的状态  $i$  和状态  $j$ ，均满足：

$$\pi(i)q_{ij} = \pi(j)q_{ji} \quad (5.54)$$

则称概率分布  $\pi$  是满足矩阵  $Q$  的一个平稳分布。

事实上，细致平稳条件的证明过程较为简单，对于任意的状态  $i$  和状态  $j$ ，均满足：



$$\sum_{i=1}^{\infty} \pi(i) q_{ij} = \sum_{i=1}^{\infty} \pi(j) q_{ji} = \pi(j) \sum_{i=1}^{\infty} q_{ji} = \pi(j)$$

由 $i$ 和 $j$ 的任意性，可以推广得：

$$\sum_{i=1}^{\infty} \pi(i) q_{ij} = \pi(j) \Leftrightarrow \pi \mathbf{Q} = \pi \quad (5.55)$$

式(5.55)表明 $\pi$ 是方程 $\pi \mathbf{Q} = \pi$ 的解，由马氏链定理可知， $\pi$ 是转移矩阵 $\mathbf{Q}$ 的平稳分布。式(5.54)也称为平衡方程（或平稳方程）。

通常情况下，转移矩阵 $\mathbf{Q}$ 是无法直接获取的，对于要模拟的概率分布 $p(x)$ ，可以首先假设有一个简单的转移矩阵 $\mathbf{Q}'$ （比如均匀的转移概率），记 $q'_{ij} = q'(x_{t+1} = j | x_t = i)$ 。

一般情况下， $p(i)q'_{ij} \neq p(j)q'_{ji}$ ，也就是不满足细致平稳条件，为此，引入一个辅助矩阵 $\mathbf{A} = \{a_{ij}\}$ ，满足：

$$p(i)q'_{ij}a_{ij} = p(j)q'_{ji}a_{ji} \quad (5.56)$$

$a_{ij}$ 也称为接受率，表示以 $a_{ij}$ 的概率接受从状态 $x_t = i$ 转移到 $x_{t+1} = j$ ；否则，以 $(1 - a_{ij})$ 的概率状态保持不变，也就是从状态 $x_t = i$ 转移到 $x_{t+1} = i$ 。对比式(5.54)与式(5.56)，不难发现：

- 对于 $i \neq j$ ，有 $q_{ij} = q'_{ij}a_{ij}$ 成立。
- 对于 $i = j$ ，有 $q_{ii} = q'_{ii} + \sum_{i \neq j} q'_{ij} * (1 - a_{ij})$ 成立。

现在来考察式(5.56)，化简得：

$$\frac{a_{ij}}{a_{ji}} = \frac{p(j)q'_{ji}}{p(i)q'_{ij}} \quad (5.57)$$

令 $\frac{p(j)q'_{ji}}{p(i)q'_{ij}} < 1$ ，可以得到接受率的计算表达式为：

$$a_{ij} = \min \left[ 1, \frac{p(j)q'_{ji}}{p(i)q'_{ij}} \right] \quad (5.58)$$

经过前面的准备工作，可以得到完整的MCMC采样算法，该算法也被称为Metropolis Hastings算法，如图5.50所示。

算法5.4 Metropolis Hastings算法

1. 初始化初始状态 $x_0$
2. 对 $t = 0, 1, \dots, n, \dots$ 循环执行下面的操作
  - 2.1 第 $t$ 时刻，马氏链的状态为 $X_t = x_t$ ，采样 $y \sim p(X_{t+1} = x | X_t = x_t)$
  - 2.2 从均匀分布中采样 $u \sim \text{Uniform}[0, 1]$
  - 2.3 若 $u < a_{ij} = \min \left[ 1, \frac{p(j)q'_{ji}}{p(i)q_{ij}} \right]$ ，则接受转移 $p(X_{t+1} = x | X_t = x_t)$ ，则 $X_{t+1} = x$
  - 2.4 否则，不接受转移，则 $X_{t+1} = x_t$

图5.50 Metropolis Hastings算法流程

### 5.7.4 Gibbs采样

5.7.3节介绍了马尔科夫链理论及MCMC采样算法，为从目标分布 $p(x)$ 中生成样本数据提供了一般性的框架，本节把这种一般性框架应用到概率图模型的推断中，也就是我们期望能够得到形如式(5.28)中 $P(Q|E = e)$ 的条件查询样本，即 $P(Q|E = e)$ 就是我们的目标分布。

由MH算法可知，我们期望能找到一个合适的状态转移矩阵 $T$ ，使得其稳定分布恰好为 $P(Q|E = e)$ ，查询变量集 $Q$ 一般是一个多维随机变量集，记为 $Q = (Q_1, Q_2, \dots, Q_k)$ 。

首先初始化初始状态 $Q = (Q_1 = q_1, Q_2 = q_2, \dots, Q_k = q_k)$ ，然后固定 $(Q_2 = q_2, \dots, Q_k = q_k)$ ，则有下面的等式成立。

$$\begin{aligned}
 & P(Q_1 = q_j, Q_2 = q_2, \dots, Q_k = q_k, E = e) \times P(Q_1 = q_i | Q_2 = q_2, \dots, Q_k = q_k, E = e) \\
 &= P(Q_2 = q_2, \dots, Q_k = q_k, E = e) \times P(Q_1 = q_j | Q_2 = q_2, \dots, Q_k = q_k, E = e) \\
 &\quad \times P(Q_1 = q_i | Q_2 = q_2, \dots, Q_k = q_k, E = e) \quad (5.59)
 \end{aligned}$$

同理：

$$\begin{aligned}
 & P(Q_1 = q_i, Q_2 = q_2, \dots, Q_k = q_k, E = e) \times P(Q_1 = q_j | Q_2 = q_2, \dots, Q_k = q_k, E = e) \\
 &= P(Q_2 = q_2, \dots, Q_k = q_k, E = e) \times P(Q_1 = q_i | Q_2 = q_2, \dots, Q_k = q_k, E = e) \\
 &\quad \times P(Q_1 = q_j | Q_2 = q_2, \dots, Q_k = q_k, E = e) \quad (5.60)
 \end{aligned}$$

式(5.59)和式(5.60)的右边等式均相等，把两者相结合，可得：

$$\begin{aligned}
 & P(Q_1 = q_j, Q_2 = q_2, \dots, Q_k = q_k, E = e) \times P(Q_1 = q_i | Q_2 = q_2, \dots, Q_k = q_k, E = e) \\
 &= P(Q_1 = q_i, Q_2 = q_2, \dots, Q_k = q_k, E = e)
 \end{aligned}$$

$$\times P(Q_1 = q_j | Q_2 = q_2, \dots, Q_k = q_k, E = e) \quad (5.61)$$

为了方便讨论, 用符号  $X = x$  来表示  $(Q_2 = q_2, \dots, Q_k = q_k, E = e)$ , 也就是说  $X$  和  $x$  满足:

$$X = (Q_2, Q_3, \dots, Q_k, E), \quad x = (q_2, q_3, \dots, q_k, e)$$

把式(5.61)简化为:

$$\begin{aligned} & P(Q_1 = q_j, X = x) \times P(Q_1 = q_i | X = x) \\ &= P(Q_1 = q_i, X = x) \times P(Q_1 = q_j | X = x) \end{aligned} \quad (5.62)$$

式(5.62)就是细致平稳条件的平衡方程, 也就是说当我们固定  $Q_2, Q_3, \dots, Q_k$ , 仅考虑单变量  $Q_1$  时,  $P(Q_1 | X = x)$  的平稳分布恰好是  $P(Q_1, X = x)$ 。基于这种思想, 可以固定变量集  $\{Q_1, Q_2, \dots, Q_{i-1}, Q_{i+1}, \dots, Q_k\}$ 。仅考虑单变量  $Q_i$ , 同样可以得出  $P(Q_i | X = x)$  的平稳分布恰好是  $P(Q_i, X = x)$  的结论。这样把多维空间的采样问题分解为单变量空间的 MCMC 采样。Gibbs 采样算法的算法流程, 如图 5.51 所示。

---

**算法 5.5 对  $n$  维空间分布  $P$  执行 Gibbs 采样算法**

---

1. 初始化初始状态  $x^0 = (x_1^0, x_2^0, \dots, x_n^0)$
  2. 对  $t = 0, 1, \dots, n, \dots$ , 循环执行下面的操作
    - 2.1  $x_1^{t+1} \sim P(x_1 | x_2^t, x_3^t, \dots, x_n^t)$
    - 2.2  $x_2^{t+1} \sim P(x_2 | x_1^{t+1}, x_3^t, \dots, x_n^t)$
    - .....
    - 2.n  $x_n^{t+1} \sim P(x_n | x_1^{t+1}, x_2^{t+1}, \dots, x_{n-1}^{t+1})$
- 

图 5.51 Gibbs 采样算法流程

图 5.52 展示了一个 Gibbs 采样的示例流程。

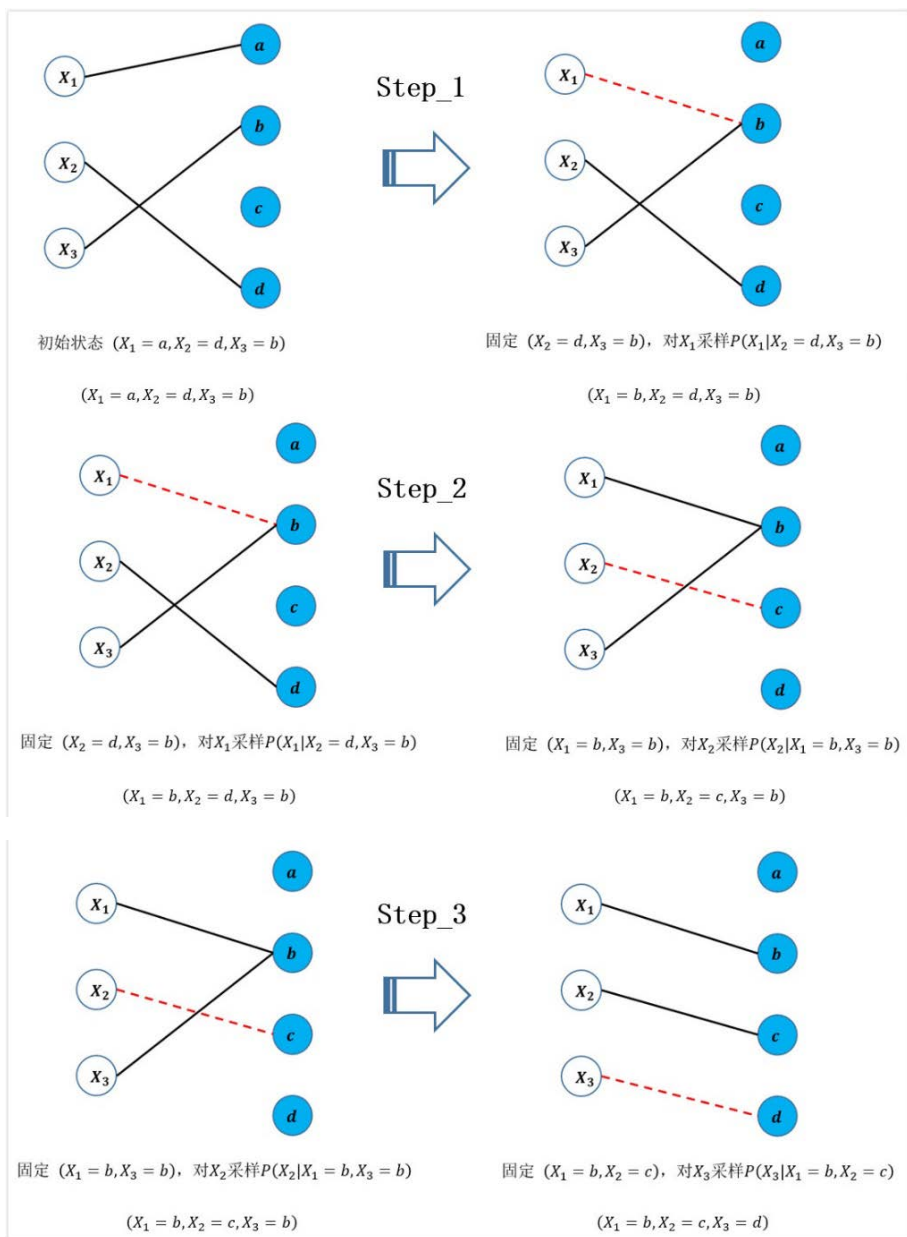


图5.52 一个Gibbs采样的示例图

采样流程：设目标分布为 $P$ ，我们的目标是得到 $P$ 的生成样本。利用Gibbs采样公式，设初始状态为 $(X_1 = a, X_2 = d, X_3 = b)$ 。固定 $X_2$ 和 $X_3$ ，对 $X_1$ 进行采样，采样使用的转移分布 $P(X_1|X_2 = d, X_3 = b)$ ，设 $P(X_1 = b|X_2 = d, X_3 = b)$ 的取值最大，得到 $X_1$ 的

状态转移结果为 $X_1 = b$ 。同理，固定 $X_1$ 和 $X_3$ ，对 $X_2$ 进行采样，采样使用的转移分布为 $P(X_2|X_1 = b, X_3 = b)$ ，设 $P(X_2 = c|X_1 = b, X_3 = b)$ 的取值最大，得到 $X_2$ 的状态转移结果为 $X_2 = c$ 。固定 $X_1$ 和 $X_2$ ，对 $X_3$ 进行采样，采样使用的转移分布为 $P(X_3|X_1 = b, X_2 = c)$ ，设 $P(X_3 = d|X_1 = b, X_2 = c)$ 的取值最大，得到 $X_3$ 的状态转移结果为 $X_3 = d$ 。这样，经过一次完整的Gibbs链转移，状态从 $(X_1 = a, X_2 = d, X_3 = b)$ 变为 $(X_1 = b, X_2 = c, X_3 = d)$ 。

## 5.8 参数学习

本节讨论PGM的最后一个模块：参数学习。到目前为止，前面讨论的问题都隐含一个假设前提，那就是假设模型参数是已知的。但在现实生活中，模型的参数取值通常是不确定的，这就要求我们采用合理的算法去估计模型的参数。

参数学习的形式化定义为，记可观察的训练数据集为 $D = \{D_1, D_2, \dots, D_n\}$ ，模型的参数为 $\theta$ ，如何由观察到的训练数据 $D$ ，来得到合理的模型参数 $\theta$ 的过程称为**参数学习**。PGM的学习理论可以区分为结构确定的参数学习和结构不确定的参数学习，本节仅讨论在模型结构已知的前提下，如何有效地学习模型的参数。

### 5.8.1 最大似然估计

当模型的结构已知，且所有变量都是可被观察时，这是参数学习最简单的形式。若模型已知，也就是说模型间各个变量间的依赖关系 $P$ 已经确定。通常使用最大似然估计（Maximum Likelihood Estimation, MLE）就能够学习到模型参数，其一般过程如下。

由样本数据，首先写出似然函数：

$$L(\theta|D) = \prod_{i=1}^n P(D_i|\theta) \quad (5.63)$$

一旦定义了似然函数，就可以利用最大似然估计来选择参数值，最大似然估计的形式化定义为：

$$\hat{\theta} = \operatorname{argmax}_{\theta} \ln(L(\theta|D)) = \operatorname{argmax}_{\theta} \sum_{i=1}^n (\ln P(D_i|\theta)) \quad (5.64)$$

通常由第7介绍的最优化方法，如梯度下降等，可以轻松求解到式(5.64)参数 $\theta$ 的

最优解 $\hat{\theta}$ 。

### 5.8.2 期望最大化算法

当模型中既含有观察变量，又含有隐藏变量时，如何有效学习模型的参数。令 $Y$ 表示可观察变量的集合， $Z$ 表示隐藏变量的集合， $\theta$ 表示模型参数集合，那么我们的目标是最大化对数似然函数：

$$\hat{\theta} = \operatorname{argmax}_{\theta} \ln(L(\theta|Y, Z)) = \operatorname{argmax}_{\theta} \ln(P(Y, Z|\theta)) \quad (5.65)$$

由于 $Z$ 是隐变量，我们没有办法观察到 $Z$ 的值，如果采用最大似然估计的思想，我们可以把所有 $Z$ 的可能取值进行累加，即(5.65)式可以写成：

$$\hat{\theta} = \operatorname{argmax}_{\theta} \ln(P(Y|\theta)) = \operatorname{argmax}_{\theta} \sum_Z \ln(P(Y, Z|\theta)) \quad (5.66)$$

这样将式(5.65)改写为只含有可观察变量集 $Y$ 的函数，式(5.66)也是对含有隐变量的模型进行参数学习的目标函数，但式(5.66)的极大化过程是非常困难的，这主要是由于对隐变量 $Z$ 的求和的项数将随着隐变量的数目增长而呈现出指数级增长的趋势。

期望最大化算法(Expectation Maximization Algorithm, 简称EM), 是由Dempster等人<sup>[8]</sup>在1977年提出的一种参数学习方法，专门用于求解含有隐变量的概率模型参数估计，EM算法并不是直接对目标函数求导得到解析解，而是通过迭代的思想来得到最优解。EM算法由E步和M步组成，下面分别讲解。

第1步：选择初始值参数 $\theta^0$ 。

第2步：E（期望）操作，假设 $\theta^t$ 为第 $t$ 次迭代后的参数估计值，那么在第 $(t+1)$ 次迭代，由给定的观察变量 $Y$ 和当前的参数估计 $\theta^t$ 推断出 $Z$ 的条件分布为 $P(Z|Y, \theta)$ ，并由此条件分布得到对数似然函数关于 $Z$ 的期望，该期望值被称为 $Q$ 函数，满足：

$$Q(\theta, \theta^t) = \sum_Z P(Z|Y, \theta^t) \times \ln(L(\theta|Y, Z)) = \sum_Z P(Z|Y, \theta^t) \times \ln(P(Y, Z|\theta)) \quad (5.67)$$

第3步：M（最大化）操作，以式(5.67)作为目标函数，寻找最优参数 $\hat{\theta}$ ，从而最大化期望似然函数，即：

$$\hat{\theta} = \operatorname{argmax}_{\theta} Q(\theta, \theta^t) \quad (5.68)$$

重复第2步和第3步的过程，当满足迭代的终止条件：

$$\|\theta^{t+1} - \theta^t\| < \varepsilon \quad \text{或} \quad \|Q(\theta^{t+1}, \theta^t) - Q(\theta^t, \theta^t)\| < \varepsilon$$

其中 $\varepsilon$ 是一个很小的数，终止迭代，最终得到参数 $\theta$ 的最优解。EM算法的整个计算过程不是直接对目标函数 $L(Y, \theta) = \ln(P(Y|\theta))$ 进行求解，而是通过构造Q函数，每一步通过最大化Q函数来获得参数的最优值。EM算法的正确性主要是通过下面的定理来保证。

**定理：**由EM算法，可以得到参数的估计序列 $\theta^t$ ， $t = 1, 2, \dots$ ，其对应的似然函数序列为 $P(Y|\theta^t)$ ，序列 $P(Y|\theta^t)$ 满足单调递增的特点，即：

$$P(Y|\theta^{t+1}) \geq P(Y|\theta^t) \quad (5.69)$$

证明：由于对数似然函数 $\ln(P(Y|\theta))$ 与似然函数 $P(Y|\theta)$ 具有相同的单调性，故考察 $\ln(P(Y|\theta))$ ，由 $P(\theta) \times P(Y|\theta) \times P(Z|Y, \theta) = P(\theta) \times P(Y, Z|\theta)$ ，可得：

$$\ln(P(Y|\theta)) = \ln(P(Y, Z|\theta)) - \ln(P(Z|Y, \theta)) \quad (5.70)$$

令：

$$Q(\theta, \theta^t) = \sum_Z P(Z|Y, \theta^t) \times \ln(P(Y, Z|\theta)) \quad (5.71)$$

$$R(\theta, \theta^t) = \sum_Z P(Z|Y, \theta^t) \times \ln(P(Z|Y, \theta)) \quad (5.72)$$

即有：

$$\begin{aligned} \ln(P(Y|\theta)) &= \ln(P(Y|\theta)) \times \sum_Z P(Z|Y, \theta^t) = \sum_Z (P(Z|Y, \theta^t) \times \ln(P(Y|\theta))) \\ &= \sum_Z (P(Z|Y, \theta^t) \times (\ln(P(Y, Z|\theta)) - \ln(P(Z|Y, \theta)))) \\ &= \sum_Z (P(Z|Y, \theta^t) \times \ln(P(Y, Z|\theta))) - \sum_Z (P(Z|Y, \theta^t) \times \ln(P(Z|Y, \theta))) \\ &= Q(\theta, \theta^t) - R(\theta, \theta^t) \end{aligned} \quad (5.73)$$

把 $\theta = \theta^t$ 和 $\theta = \theta^{t+1}$ 分别代入式(5.73)可得：

$$\begin{aligned} &\ln(P(Y|\theta^{t+1})) - \ln(P(Y|\theta^t)) \\ &= [Q(\theta^{t+1}, \theta^t) - Q(\theta^t, \theta^t)] - [R(\theta^{t+1}, \theta^t) - R(\theta^t, \theta^t)] \end{aligned} \quad (5.74)$$

其中 $Q$ 由EM算法的最大化操作保证了 $Q(\theta^{t+1}, \theta^t) - Q(\theta^t, \theta^t) \geq 0$ ，下面考察 $R(\theta^{t+1}, \theta^t) - R(\theta^t, \theta^t)$ ，由Jensen不等式可得：

$$\begin{aligned}
R(\theta^{t+1}, \theta^t) - R(\theta^t, \theta^t) &= \sum_Z \left( P(Z|Y, \theta^t) \times \ln \frac{P(Z|Y, \theta^{t+1})}{P(Z|Y, \theta^t)} \right) \\
&\leq \ln \left( \sum_Z \left( P(Z|Y, \theta^t) \times \frac{P(Z|Y, \theta^{t+1})}{P(Z|Y, \theta^t)} \right) \right) \\
&= \ln \left( \sum_Z P(Z|Y, \theta^{t+1}) \right) = 0
\end{aligned} \tag{5.75}$$

把式(5.75)代入式(5.74)可得 $\ln(P(Y|\theta^{t+1})) - \ln(P(Y|\theta^t)) \geq 0$ 成立， $\ln(P(Y|\theta))$ 满足单调递增的性质，证明的开头已经知道 $P(Y|\theta)$ 与 $\ln(P(Y|\theta))$ 具有相同的单调性，故：

$$P(Y|\theta^{t+1}) \geq P(Y|\theta^t)$$

成立，定理证毕。

由文献[9]可知，在绝大多数情况下， $L(Y, \theta) = \ln(P(Y|\theta))$ 是有界的，故使用EM算法能保证收敛到最优解。但需要注意的是EM算法只能保证得到局部最优解，且EM算法受到初始值的影响，因此，初始值的选择对于EM算法的结果起到非常关键的作用。

## 5.9 小结

本章对PGM的理论进行了详细讲解，包括表示理论、推断理论和学习理论的相关知识，但正如本章开篇所描述，PGM的体系和框架非常庞大，限于本书的篇幅，我们没有对PGM的所有知识点进行全面的分析。下面简要概括与本章知识点相关的更深入的方法和理论，读者可以查阅相关的文献或者书籍来进行更深入的研究分析。

- 表示理论：本章讲解了PGM两种重要的表示，分别是贝叶斯网络和马尔科夫网络，除此之外，混合图模型（即图中既包含有向边也包含无向边）和基于时序的模型也是PGM很重要的两种模型表示。
- 推断理论：本章概述了包括变量消除和信念传播在内的精准推断方法，以及基于MCMC采样的近似推断方法，其他常用的近似推断方法还包括变分推断（Variational Inference）和松散的信念传播（Loopy Belief Propagation）。
- 学习理论：本章仅探讨了结构已经确定的情况下，如何对概率模型的参数进行学习。事实上，PGM的参数学习在很多情况下都是结构不确定的，也就是预先不知道模型的表示。结构不确定的参数学习是一个非常困难的过程，一般采用的策略是基于得分的学习策略（score-based Learning），典型的算法包括启发



式搜索（Heuristic Search）和结构化EM算法（结构化EM适用于结构不确定，且含有隐变量的情形）。

### 参考文献：

---

- [1] Daphne Koller, Nir Friedman. Probabilistic Graphical Models, Principles and Techniques. The MIT Press. ISBN-13: 978-0262013192.
- [2] Andrew Y. Ng, Michael I. Jordan. (2001). On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes. In NIPS (pp. 841–848).
- [3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. Introduction to Algorithms, third edition. The MIT Press.
- [4] Richard A. Brualdi. Introductory Combinatorics, 5th Edition.
- [5] D.H.Lehmer. on Large-Scale Digital Calculating Machinery. Cambridge, Mass: Harvard University Press. 141-146. 1951.
- [6] Donald E.Knuth. The Art of Computer Programming, Vol 2: Seminumerical Algorithms. Third Edition.
- [7] Bishop M. Pattern Recognition and Machine Learning. Springer-Verlag. 2006.
- [8] Dempster, A.P.; Laird, N.M.; Rubin, D.B. (1977). "Maximum Likelihood from Incomplete Data via the EM Algorithm". Journal of the Royal Statistical Society, Series B. 39 (1): 1–38. JSTOR 2984875. MR 0501537.
- [9] Radford N, Geoffrey H, Jordan MI. A view of the EM algorithm that justifies incremental, sparse , and other variants. Learning in Graphical Models. Cambridge, MA: MIT Press, 1999, 355-368.

# 6

## 机器学习基础

机器学习是人工智能领域的重要分支，被认为是最能体现机器智能的一门学科。从20世纪80年代中期开始，统计机器学习逐渐成为了机器学习的主流发展方向，并使得人工智能从早期纯粹的模型和理论研究发展为可以解决现实生活问题的应用研究。近年来，随着大数据时代的来临，以及高性能计算平台的发展，使得机器学习的发展取得了很多理论和应用的突破，并在此基础上，催生了很多新的理论，这其中深度学习，就是当前机器学习最热门的研究领域。也就是说深度学习并不是一门全新的学科，它是机器学习的一个分支流派，很多理论和基础其实都来源于统计机器学习，与统计机器学习相对应，人们也把以神经网络为代表的深度学习称为连接主义（connectionism）机器学习。

本章将介绍机器学习，尤其是统计机器学习的相关知识点。机器学习要解决的问题是，基于数据构建合理的统计模型，并利用该模型对数据进行分析和预测。一般来说，统计机器学习可以分为监督学习、无监督学习和强化学习。本章主要介绍几种常用的监督学习和无监督学习算法。

机器学习的体系和架构非常庞大，限于本书的篇幅和写作目的，本章只对几个常见的算法理论进行简要介绍，如果读者想深入研究机器学习的相关知识，当前有很多优秀的国内外机器学习相关文献，如参考文献[1,2,3,4]，读者可自行查阅。

## 6.1 线性模型

线性模型 (linear model), 是机器学习中的一类算法总称, 其形式化定义为: 通过给定的样本数据集  $D$ , 线性模型试图学习到这样的一个模型, 使得对于任意的输入特征向量  $\mathbf{x} = (x_1, x_2, \dots, x_n)^T$ , 模型的预测输出  $f(\mathbf{x})$  能够表示为输入特征向量  $\mathbf{x}$  的线性函数, 即满足:

$$f(\mathbf{x}) = w_1x_1 + w_2x_2 + \dots + w_nx_n + b \quad (6.1)$$

也可以将式(6.1)写成矩阵的形式:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b \quad (6.2)$$

其中,  $\mathbf{w} = (w_1, w_2, \dots, w_n)^T$  和  $b$  称为模型的参数。线性模型是机器学习中最简单、最基础的模型结构, 常常被应用于分类、回归等学习任务中。线性模型也是非线性模型的基础, 事实上, 很多非线性模型都是在线性输出结果的基础上进行非线性变换、层级叠加等操作的。因此, 本章首先介绍线性模型的相关知识。常见的线性模型包括线性回归、单层感知机和 Logistic 回归, 其中单层感知机也属于一种神经网络模型, 考虑到本书结构的连贯性, 我们将其放在第8章中讲解。

### 6.1.1 线性回归

回归 (regression) 是监督学习任务的一种, 形式化地定义为: 设给定由  $m$  个训练样本数据构成的数据集  $D$ :

$$D = \{(x^1, y^1), (x^2, y^2), \dots, (x^m, y^m)\}$$

其中,  $\mathbf{x}^i = (x_1^i, x_2^i, \dots, x_n^i)^T$  表示第  $i$  个训练数据的输入特征向量,  $y^i \in \mathbf{R}$ , 回归分析的任务是通过训练数据集  $D$  学习到一个模型  $T$ , 使得模型  $T$  能够尽量拟合训练数据集  $D$ , 并且对于新的输入数据  $\mathbf{x}$ , 应用模型  $T$  能够得到预测结果  $f(\mathbf{x})$ 。回归与分类是监督学习的两种形式, 它们的概念很接近, 唯一的区别在于, 回归的预测值是一个连续的实数, 而分类任务的预测值是离散的类别数据。

线性回归是回归学习的一种策略, 其试图通过对训练集  $D$  的学习, 使得输入  $\mathbf{x}$  和预测的输出  $f(\mathbf{x})$  之间具有形如式(6.1)的线性关系。要确定模型  $T$ , 需要确定参数  $\mathbf{w}$  和  $b$ , 那么应该如何求解参数呢? 首先需要定义一种衡量标准, 用于衡量模型的预测值  $f(\mathbf{x})$  与准确值  $y$  之间的差距, 也就是定义损失函数。在回归任务中, 常用的损失函数是均

方误差：

$$L(w, b) = \frac{1}{2} \sum_{i=1}^m (f(x^i) - y^i)^2 \quad (6.3)$$

我们的目标是要最小化式(6.3)所示的损失函数，基于均方误差来求解模型的方法也被称为最小二乘法（Ordinary Least Square Method，简称为OLS），最小二乘法的思想是寻找一个超平面，使得训练数据集 $D$ 中的所有样本点到超平面的距离之和最小，如图6.1所示，在本节的最后，将讲解最小二乘法与高斯分布的关系。

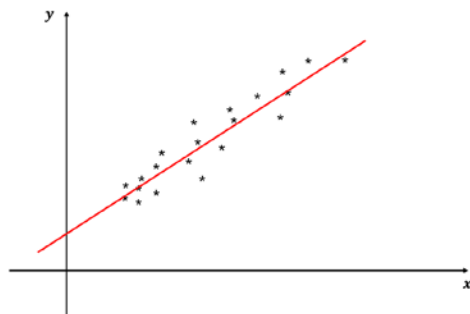


图6.1 线性回归

这样，我们将机器学习的问题转化为数值最优化问题。有关最优化的算法策略，将在第7章详细讲解。要求解式(6.3)的最小化问题，有两种常用的方法。

第一种方法，利用迭代法求解，即利用第7章介绍的各种最优化算法，例如梯度下降法等。通过这些算法，能够很方便地得到最优参数 $w^*$ 和 $b^*$ 。

第二种方法，直接利用解析法来求解，即根据极值存在的必要条件，对损失函数的参数 $w$ 和 $b$ 进行求导得到参数方程组，令参数方程组为0，将最优化问题转化为求解方程组问题。在下一章将介绍，在一般情况下，利用解析法求解最优化问题是不可行的，但对于线性回归来说，它的模型足够简单，在数据量不大且满足一定的条件下，利用解析法来求解会更加高效。

下面详细推导出解析法的结果。要利用解析法求解，首先把参数 $w$ 和 $b$ 合并，并统一使用符号 $\theta$ 来表示， $\theta$ 满足：

$$\theta = (w_1, w_2, \dots, w_n, b)^T \quad (6.4)$$

容易知道 $\theta$ 是一个 $(n+1)$ 维的向量，同时把第 $i$ 个训练数据的输入特征向量改写为：

$$\mathbf{X}^i = (x_1^i, x_2^i, \dots, x_n^i, 1)^T \quad (6.5)$$

则全体训练集的输入特征向量可用矩阵表示为：

$$\mathbf{X} = \begin{pmatrix} (X^1)^T \\ (X^2)^T \\ \dots \\ (X^m)^T \end{pmatrix} \quad (6.6)$$

$\mathbf{X}$ 是一个大小为 $m \times (n+1)$ 维的矩阵，这样得到线性回归的全体训练数据的预测输出值满足：

$$f(\mathbf{X}) = \mathbf{X}\boldsymbol{\theta} \quad (6.7)$$

对输出数据，设：

$$\mathbf{Y} = (y^1, y^2, \dots, y^m)^T \quad (6.8)$$

经过新的符合定义，可以将式(6.3)的损失函数转化为矩阵形式：

$$L(\boldsymbol{\theta}) = \frac{1}{2} (\mathbf{X}\boldsymbol{\theta} - \mathbf{Y})^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{Y}) \quad (6.9)$$

利用极值存在的必要条件，可以直接对式(6.9)的参数 $\boldsymbol{\theta}$ 求导，得：

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \frac{1}{2} (\mathbf{X}\boldsymbol{\theta} - \mathbf{Y})^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{Y}) \\ &= \frac{1}{2} \nabla_{\boldsymbol{\theta}} (\boldsymbol{\theta}^T \mathbf{X}^T \mathbf{X} \boldsymbol{\theta} - \boldsymbol{\theta}^T \mathbf{X}^T \mathbf{Y} - \mathbf{Y}^T \mathbf{X} \boldsymbol{\theta} + \mathbf{Y}^T \mathbf{Y}) \\ &= \frac{1}{2} \nabla_{\boldsymbol{\theta}} \text{tr}(\boldsymbol{\theta}^T \mathbf{X}^T \mathbf{X} \boldsymbol{\theta} - \boldsymbol{\theta}^T \mathbf{X}^T \mathbf{Y} - \mathbf{Y}^T \mathbf{X} \boldsymbol{\theta} + \mathbf{Y}^T \mathbf{Y}) \\ &= \frac{1}{2} \nabla_{\boldsymbol{\theta}} (\text{tr}(\boldsymbol{\theta}^T \mathbf{X}^T \mathbf{X} \boldsymbol{\theta}) - 2\text{tr}(\mathbf{Y}^T \mathbf{X} \boldsymbol{\theta})) \\ &= \frac{1}{2} (2\mathbf{X}^T \mathbf{X} \boldsymbol{\theta} - 2\mathbf{X}^T \mathbf{Y}) = \mathbf{X}^T \mathbf{X} \boldsymbol{\theta} - \mathbf{X}^T \mathbf{Y} \end{aligned} \quad (6.10)$$

上面的推导过程要使用到矩阵的迹运算性质，读者可以参考第3章的相关知识点，或者查阅相关的线性代数文献，这里不再详述。令式(6.10)的求导结果为0，得到参数 $\boldsymbol{\theta}$ 的表达式为：

$$\mathbf{X}^T \mathbf{X} \boldsymbol{\theta} - \mathbf{X}^T \mathbf{Y} = 0 \Rightarrow \boldsymbol{\theta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y} \quad (6.11)$$

式(6.11)就是解析法的求解公式，对于新的测试数据 $\mathbf{x}$ ，利用式(6.12)求得其预测输出为：

$$f(x) = \theta^T x \quad (6.12)$$

注意,利用式(6.11)来求解参数 $\theta$ 时,需要保证对称矩阵 $\mathbf{X}^T \mathbf{X}$ 是可逆的。如果 $\mathbf{X}^T \mathbf{X}$ 不可逆,则式(6.11)会失效。

下面考察线性回归的一种改进:局部加权线性回归(Locally Weighted Linear Regression, 简称 LWR)。线性回归是采用超平面来拟合所有的训练数据,但如果训练数据不是呈现线性的分布关系时,在大部分情况下,线性模型得到的结果都会出现欠拟合的现象,如图6.2所示。

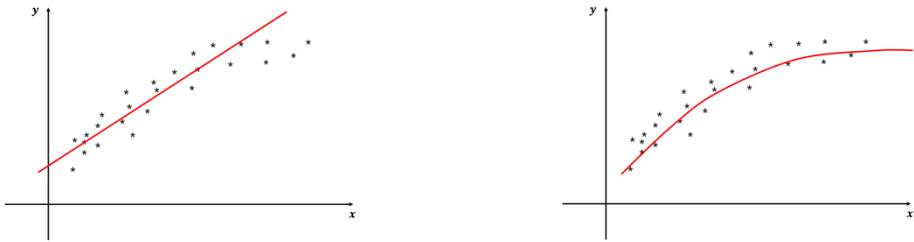


图6.2 左图:线性回归得到的模型效果并不好,出现欠拟合现象;右图,通过添加组合特征来使得拟合的效果更好,但模型也变得更复杂

从图6.2可以看出,要解决欠拟合的问题,一种方法是挖掘更多的特征,比如不同特征之间的组合,但这样做会使得模型更复杂,而且好的特征选取并不是一件简单的事情,另一种方法是通过修改线性回归,也就是马上要讲解的LWR,使得我们在不添加新特征的前提下,获得近似的效果。LWR的思想比较简单,只需要将式(6.3)的损失函数修改为:

$$L(w, b) = \frac{1}{2} \sum_{i=1}^m \mu^i (f(x^i) - y^i)^2 \quad (6.13)$$

其中,  $\mu^i$  是一个非负的权重值,一般采用指数函数的形式:

$$\mu^i = \exp\left(-\frac{(x^i - x)^2}{2\tau^2}\right) \quad (6.14)$$

其中,  $x$  表示测试数据,可以仿照式(6.10)的推导过程,求得式(6.13)的最优解参数的求解公式为:

$$\theta = (\mathbf{X}^T \boldsymbol{\psi} \mathbf{X})^{-1} \mathbf{X}^T \boldsymbol{\psi} \mathbf{Y} \quad (6.15)$$

其中,  $\boldsymbol{\psi}$  是一个对角矩阵,满足:

$$\psi_{i,i} = \mu^i = \exp\left(-\frac{(x^i - x)^2}{2\tau^2}\right) \quad (6.16)$$

从直观上来理解, LWR在线性回归的基础上, 为第 $i$ 训练数据添加权重值 $\mu^i$ , 从式(6.14)知道, 当 $x^i$ 与 $x$ 越接近时,  $\mu^i$ 的值越大(最大值为1); 相反, 当 $x^i$ 与 $x$ 的距离越远时,  $\mu^i$ 的值越小(最小值为0)。也就是说, 当使用式(6.12)来求解新数据 $x$ 的预测输出时, 离 $x$ 越近的点 $x^i$ , 它对结果的影响权重越大, 离 $x$ 越远的点, 对结果的影响越小。

当训练数据比较多的时候, LWR能够取得比较不错的效果。但利用LWR来进行预测最大的缺点是空间的开销比较大, 在线性回归模型中, 当通过训练得到参数 $\theta$ 的最优解后, 可以舍弃训练数据, 只需要保留参数 $\theta$ 的最优解, 就可以得到新数据的预测输出, 但LWR除了保留参数 $\theta$ 的最优解外, 还要保留全部的训练数据, 以求取每一个训练数据对应于新输入数据的权重值。

最后, 考察最小二乘法与高斯分布的关系。首先直接给出结论: 利用最小二乘法来拟合训练数据, 等价于把训练数据的输出看成是服从高斯分布。

第 $i$ 个训练样本的真实输出 $y^i$ 和预测输出 $f(x^i)$ 存在下面的等式关系:

$$y^i = f(x^i) + \epsilon^i = \theta^T x^i + \epsilon^i \quad (6.17)$$

其中,  $\epsilon^i$ 表示由噪音引起的误差项, 所有训练数据的误差项满足独立同分布的关系, 一般情况下,  $\epsilon^i$ 服从均值为0的高斯分布, 即:

$$p(\epsilon^i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\epsilon^i)^2}{2\sigma^2}\right) \quad (6.18)$$

把式(6.17)代入式(6.18)可得:

$$p(y^i|x^i; \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\theta^T x^i - y^i)^2}{2\sigma^2}\right) \quad (6.19)$$

式(6.19)表示, 当固定参数 $\theta$ , 给定输入数据 $x^i$ , 输出值 $y^i$ 满足均值为 $\theta^T x^i$ 的高斯分布。利用对数最大似然估计, 有:

$$\begin{aligned} L(\theta) &= \ln\left(\prod_{i=1}^m p(y^i|x^i; \theta)\right) \\ &= \sum_{i=1}^m \ln(p(y^i|x^i; \theta)) \end{aligned}$$

$$= m \times \ln \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \times \frac{1}{2} \sum_{i=1}^m (\theta^T x^i - y^i)^2 \quad (6.20)$$

其中， $m \times \ln \frac{1}{\sqrt{2\pi}\sigma}$  是常数项，因此要最大化式(6.20)，实际上等价于最小化函数  $\frac{1}{2} \sum_{i=1}^m (\theta^T x^i - y^i)^2$ ，而  $\frac{1}{2} \sum_{i=1}^m (\theta^T x^i - y^i)^2$  恰好是式(6.3)所示的损失函数，也就是说利用最小二乘法求解的线性模型，等价于数据输出值的分布服从高斯分布。

### 6.1.2 Logistic回归

6.1.1节介绍了使用线性模型来进行回归学习，本节将介绍如何使用线性模型来进行分类学习。使用线性模型来分类，最常用的算法是Logistic回归，与回归任务不同，分类的结果输出是有限的离散值，为了后面讲解的方便，以二元分类为例，结果输出要么为0，要么为1，即  $y \in \{0,1\}$ 。利用线性模型来进行分类学习，其基本思想是在空间中构造一个合理的超平面，把空间区域划分为两个子空间，每一种类别数据都在平面的某一侧，如图6.3所示。

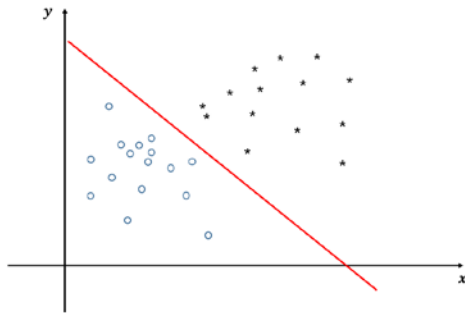


图6.3 线性模型分类

不能直接按式(6.1)的结果来分类，因为直接使用公式得到的是实数值，需要将实数值规约为0或1。最理想的是阶跃函数，如图6.4所示。

$$f(x) = \begin{cases} 0 & w^T x + b \leq 0 \\ 1 & w^T x + b > 0 \end{cases} \quad (6.21)$$



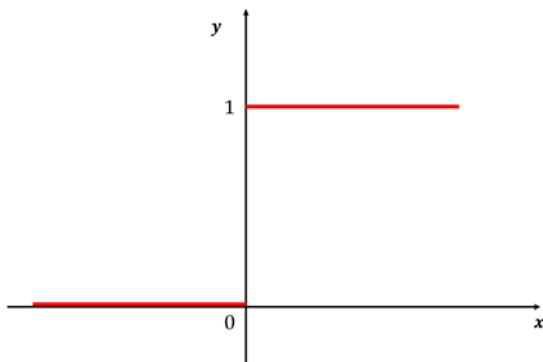


图6.4 阶跃函数

但阶跃函数具有不连续、不可导的特点。为此，可以通过阶跃函数的平滑版本，即sigmoid函数来为我们实现：

$$f(x) = \frac{1}{1 + e^{-(w^T x + b)}} \quad (6.22)$$

sigmoid函数具有很多优秀的性质：首先，它将输入数据压缩为0到1的范围内，得到的结果不是二值输出，而是一个概率值，通过这个数值，可以查看输入数据分别属于0类或属于1类的概率：

$$p(y = 1|x) = f(x) = \frac{e^{w^T x + b}}{1 + e^{w^T x + b}} \quad (6.23)$$

$$p(y = 0|x) = 1 - f(x) = \frac{1}{1 + e^{w^T x + b}} \quad (6.24)$$

其次，sigmoid函数是一个可导的函数，且其导数的形式非常简单：

$$\nabla f(x) = f(x) \times (1 - f(x)) \quad (6.25)$$

与线性回归一样，要确定Logistic回归模型，需要确定参数 $w$ 和 $b$ ，Logistic回归的损失函数采用对数最大似然损失，即满足：

$$\begin{aligned} L(w, b) &= \ln \left( \prod_{i=1}^m p(y^i | x^i; w, b) \right) \\ &= \sum_{i=1}^m \ln (p(y^i | x^i; w, b)) \end{aligned} \quad (6.26)$$

其中，由于在二元分类中， $y^i$ 的可能取值只能为0或1，故 $p(y^i | x^i; w, b)$ 可以写成

更紧凑的表达形式：

$$p(y^i|x^i;w,b) = \left(f(x^i)\right)^{y^i} \left(1-f(x^i)\right)^{1-y^i} \quad (6.27)$$

将式(6.27)代入式(6.26)，并将最大化问题转化为最小化问题，得：

$$L(w,b) = -\sum_{i=1}^m \left( y^i \times \ln \left( f(x^i) \right) + (1-y^i) \times \ln \left( 1-f(x^i) \right) \right) \quad (6.28)$$

这样将Logistic回归转化为最优化问题，目标是最小化式(6.28)所示的损失函数。

### 6.1.3 广义的线性模型

前面讲解了两种线性模型，事实上，它们都是广义线性模型（Generalized Linear Models，简称GLM）的特殊形式，本节探讨如何构建GLM模型来进行任务学习<sup>[4,5]</sup>。

对任意给定的由 $m$ 个训练数据构成的数据集 $D$ ：

$$D = \{(x^1, y^1), (x^2, y^2), \dots, (x^m, y^m)\}$$

假设数据的输出值 $y$ 的分布服从式(6.29)所示的指数分布，那么对于新的输入数据 $x$ ，其预测输出为 $f(x) = E(p(y|x;\eta))$ 。

$$p(y|x;\eta) = b(y) \times \exp(\eta^T T(y) - a(\eta)) \quad (6.29)$$

其中， $T(y) = y$ ， $\eta = \theta^T x$ 。

考察线性回归模型，6.1.1节提到，利用最小二乘法求解时，数据的输出分布服从高斯分布：

$$p(y|x;\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\theta^T x - y)^2}{2\sigma^2}\right)$$

可以将其化简为式(6.29)的指数分布表示：

$$p(y|x;\theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{y^2}{2}\right) \times \exp\left((\theta^T x)y - \frac{(\theta^T x)^2}{2}\right) \quad (6.30)$$

因此，最小二乘法的概率模型满足式(6.29)的表示，对于高斯分布，它的期望值为 $E(p(y|x;\theta)) = \theta^T x$ ，线性模型的预测输出值为 $f(x) = \theta^T x$ ，与式(6.12)相吻合。

再来讨论Logistic回归，Logistic回归的输出服从伯努利分布，即0-1分布，令：

$$p(y = 1|x; \theta) = \phi$$

将伯努利分布转化为指数分布的形式：

$$\begin{aligned} p(y|x; \theta) &= \phi^y \times (1 - \phi)^{1-y} \\ &= \exp(\ln(\phi^y \times (1 - \phi)^{1-y})) \\ &= \exp(y \times \ln(\phi) + (1 - y) \times \ln(1 - \phi)) \\ &= \exp\left(y \times \ln\left(\frac{\phi}{1 - \phi}\right) + \ln(1 - \phi)\right) \end{aligned} \quad (6.31)$$

将式(6.31)与式(6.29)进行比较，有：

$$\eta = \theta^T x = \ln\left(\frac{\phi}{1 - \phi}\right) \Rightarrow \phi = \frac{1}{1 + e^{-\theta^T x}} \quad (6.32)$$

对于伯努利分布，有 $E(p(y|x; \theta)) = p(y = 1|x; \theta) = \phi$ 成立，根据广义线性模型的结论，可以利用 $\phi$ 来预测新数据的输出。而 $\phi$ 的表达式式(6.32)与式(6.23)也完全一致。

## 6.2 支持向量机

支持向量机 (support vector machine, 简称SVM), 由Corinna Cortes和Vapnik于1993年提出, 并于1995年发表<sup>[6]</sup>, 是机器学习中极具代表性的算法。首先回顾二元分类问题, 假设数据是线性可分的, 那么利用线性模型来构建分类器的一般做法是: 构建一个合理的超平面, 设超平面方程可以表示为:

$$w^T x + b = 0 \quad (6.33)$$

利用该超平面将不同的类别数据分隔开来, 比如当 $w^T x + b < 0$ 时, 把数据 $x$ 归为-1类; 当 $w^T x + b \geq 0$ 时, 把数据 $x$ 归为+1类。但满足这样的超平面可能有很多, 如图6.5所示。

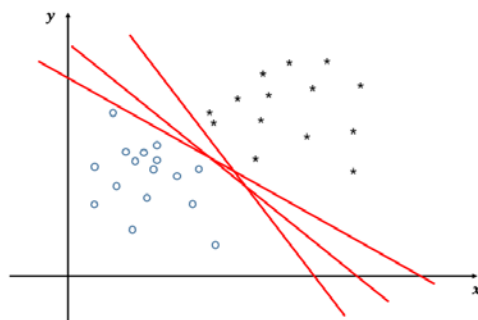


图6.5 可以构建多个超平面将不同的类别数据分开

事实上，图6.5中的超平面都能满足把不同类别数据分隔开来的需求，因此，我们的任务是从中挑选最合理的超平面模型。一个好的模型，它应该具有高的鲁棒性以及优秀的泛化能力。具体来说，对于二元分类，一个好的模型应该对噪音产生的数据具有较高的容忍度。从几何的角度来看，训练数据的点离超平面越远越好，离超平面越远，我们对其的预测输出结果的置信度就越高；反之，离超平面越近，由于一些细微的变化，都可能导致数据被划分到不同的类别中，我们自然对其预测的置信度大大降低。

### 6.2.1 最优间隔分类器

本节将给出前面描述的形式化定义，也就是如何构建SVM的最优化模型。为此，首先引入两个间隔定义来衡量预测的置信度。

#### 1. 函数间隔 (functional margin)

首先，回忆上一节的Logistic回归模型。Logistic回归通过sigmoid函数来得到输入数据属于类别1的概率 $p(y = 1|x)$ ，通常来说，当 $p(y = 1|x) \geq 0.5$ 时，将输出类别1，否则，输出类别0。

现在考察这样的问题，有两个输入点 $x_1$ 和 $x_2$ ，将其代入Logistic回归模型，得到它们的输出结果分别为 $p(y = 1|x_1) = 0.55$  和  $p(y = 1|x_2) = 0.98$ ，可以看出，虽然两个输入点都被归类为类别1，但预测的可靠性却有很大的不同，显然我们会对把 $x_2$ 归类为类别1更有信心。

从上面的分析可以看出，当 $p(y = 1|x_1)$ 的值越接近于1，也就是 $w^T x + b \gg 0$ 时，对预测的输出为类别1的结果越有信心；同理，当 $p(y = 1|x_1)$ 的值越接近于0，则

$w^T x + b \ll 0$ 时，我们对预测的输出为类别0的结果越有信心。

引入函数间隔来描述这种预测的可靠性，为了使分析更加简单，从现在开始，本节后面我们考虑二元分类时，输出值的可取数值为 $\{-1, 1\}$ 。当 $w^T x + b \geq 0$ 时，输出类别1，当 $w^T x + b < 0$ 时，输出类别-1。函数间隔的定义为：

$$\hat{d}_i = y^i \times (w^T x^i + b) \quad (6.34)$$

若训练数据是线性可分的，那么 $\hat{d}_i$ 恒为非负数，当 $\hat{d}_i$ 值越大，若 $y^i = 1$ ，则 $w^T x + b \gg 0$ ，越确信数据属于类别1；若 $y^i = -1$ ，则 $w^T x + b \ll 0$ ，越确信数据属于类别-1。函数间隔能很好地反映出预测的结果置信度。

如前所述，对于给定的由 $m$ 个训练数据构成的数据集 $D$ ：

$$D = \{(x^1, y^1), (x^2, y^2), \dots, (x^m, y^m)\}$$

离超平面越近的对模型的鲁棒性要求越高，因此，通常关心的是那些离超平面最近的点，这些边界点被称为支持向量（或支撑向量，support vector），如图6.6所示。

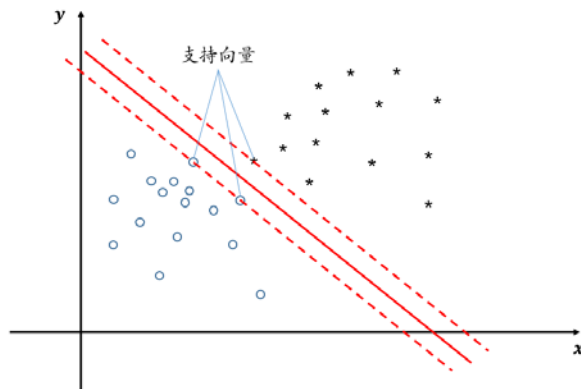


图6.6 支持向量表示离超平面最近的点

支持向量到超平面的距离是数据集 $D$ 到超平面的最短距离，形式化定义为：

$$\hat{d} = \min_{i=1,2,\dots,m} \hat{d}_i \quad (6.35)$$

## 2. 几何间隔 (geometric margin)

使用函数间隔来衡量预测的置信度有一个不足的地方，那就是当等比例同时放大参数 $w$ 和 $b$ 时，超平面的位置实际上并没有任何变化，而且点到超平面的距离也没有变

化，但函数间隔却变大了，为了克服这个缺点，引入了几何间隔的定义：

$$d_i = \frac{y^i \times (w^T x^i + b)}{\|w\|} \quad (6.36)$$

从式(6.36)可知，几何间隔实际上就是点到超平面的欧氏距离，如图6.7所示。

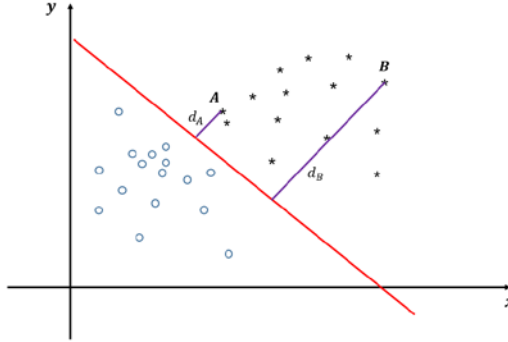


图6.7 几何间隔

几何间隔的意义与函数间隔一样，当几何间隔越大，其预测的结果越可靠，例如图6.7中的点A和点B，显然点B属于类别1的概率要远远大于点A。

从函数间隔与几何间隔的定义可以看出，两者之间相差一个 $\|w\|$ 因子，即：

$$d_i = \frac{\hat{d}_i}{\|w\|} \quad (6.37)$$

与函数间隔一样，重点关注支持向量的几何间隔，定义为：

$$d = \min_{i=1,2,\dots,m} d_i \quad (6.38)$$

对于给定的训练数据集 $D$ ，我们的目标是希望间隔越大越好，尤其是我们希望那些边界点，即支持向量的间隔离超平面越大越好，为此，得到了下面的带约束的最优化问题：

$$\begin{aligned} & \max_{w,b} \hat{d} \\ & \text{s.t. } y^i \times (w^T x^i + b) \geq \hat{d} \quad i = 1, 2, \dots, m \\ & \|w\| = 1 \end{aligned} \quad (6.39)$$

但由于约束条件 $\|w\| = 1$ 非凸不可导，我们尝试修改式(6.39)的约束优化问题，将目标函数从函数间隔的最大值过渡到几何间隔的最大值，如下式所示：

$$\begin{aligned} \max_{w,b} \quad & \frac{\hat{d}}{\|w\|} \\ \text{s.t.} \quad & y^i \times (w^T x^i + b) \geq \hat{d} \quad i = 1, 2, \dots, m \end{aligned} \quad (6.40)$$

对式(6.34)的左右两边同时乘于一个相同的数，等式不变，因此我们总可以将支持向量到超平面的函数间隔值设置为1，也就是 $\hat{d} = 1$ ，同时，通过将最大化问题转化为求最小化问题，使得目标函数变为一个二次的凸函数，凸函数的最优化问题远比非凸函数简单，且能保证获得最优解。通过上面的步骤，得到下面的约束最优化问题：

$$\begin{aligned} \max_{w,b} \quad & \frac{1}{2} \|w\|^2 \\ \text{s.t.} \quad & y^i \times (w^T x^i + b) \geq 1 \quad i = 1, 2, \dots, m \end{aligned} \quad (6.41)$$

式(6.41)就是要求解的SVM最优化模型，由其最优解得到的分类器，成为最优间隔分类器 (optimal margin classifier)，式(6.41)也被称为SVM最优化的基本型。

## 6.2.2 对偶问题

本节考察(6.41)式的原始最优化问题的对偶问题。事实上，原始最优化问题是一个带线性约束条件，且目标函数是二次函数的凸优化问题，通过数值最优化策略已经可以求解出最优值，那么为什么还需要考虑其对偶问题呢？这主要是基于两方面的考虑：一是对偶问题的求解比直接求解原始问题的效率更高；二是对偶问题能更方便引入核函数。

求解对偶问题，需要用到拉格朗日乘子法和KKT条件的相关知识，我们将在下一章讲解，本节将不加验证，直接应用其结论，对式(6.41)的约束优化模型构建拉格朗日函数，引入拉格朗日乘子 $a_i, i = 1, 2, \dots, m$ ，得：

$$L(w, b, a) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^m a_i [y^i \times (w^T x^i + b) - 1] \quad (6.42)$$

将拉格朗日函数分别对 $w$ 和 $b$ 求偏导，并令偏导方程为0，得：

$$\frac{\partial L(w, b, a)}{\partial w} = w - \sum_{i=1}^m a_i y^i x^i = 0 \Rightarrow w = \sum_{i=1}^m a_i y^i x^i \quad (6.43)$$

$$\frac{\partial L(w, b, a)}{\partial b} = \sum_{i=1}^m a_i y^i = 0 \quad (6.44)$$

将式(6.43)和式(6.44)重新代入式(6.42)，消去参数 $w$ 和 $b$ ，得到只含有 $a$ 的函数：

$$L(a) = \sum_{i=1}^m a_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m y^i y^j a_i a_j (x^i)^T x^j \quad (6.45)$$

这样将式(6.41)的原始问题转化为下面的对偶问题：

$$\begin{aligned} \min_a \quad & \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m y^i y^j a_i a_j (x^i)^T x^j - \sum_{i=1}^m a_i \\ \text{s.t.} \quad & \sum_{i=1}^m a_i y^i = 0 \quad i = 1, 2, \dots, m \\ & a_i \geq 0 \quad i = 1, 2, \dots, m \end{aligned} \quad (6.46)$$

要求解式(6.46)所示的对偶问题，可以利用常规的二次规划最优化来求解，但对偶问题存在更加高效的解法。1998年，Platt发明了一种序列最小优化（Sequential Minimal Optimization，简称SMO）的算法，极大提升了SVM的训练效率，读者可以参考文献[7]，查看其详细的推导过程。

当求解出式(6.46)的最优参数 $a^*$ 后，将 $a^*$ 代入式(6.43)即可求出原始问题的参数 $w$ 的最优解 $w^*$ ，满足 $w^* = \sum_{i=1}^m a_i^* y^i x^i$ ，对于参数 $b$ ，其最优解通过下式求得：

$$b^* = - \frac{\max_{y^i=-1} (w^*)^T x^i + \min_{y^i=1} (w^*)^T x^i}{2} \quad (6.47)$$

利用最优值 $a^*$ 、 $w^*$ 、 $b^*$ ，可以得到分类器的最终模型为：

$$f(x) = w^T x + b = \left( \sum_{i=1}^m a_i^* y^i x^i \right) x + b^* = \sum_{i=1}^m a_i^* y^i x^i x + b^* \quad (6.48)$$

### 6.2.3 核函数

本节的最后将介绍核函数的作用，在前面的分析中，我们都假设数据是线性可分的，但在很多实际应用中，遇到更多的是线性不可分的数据，核函数的作用是通过将线性不可分的输入特征向量映射到高维空间中，使得映射后的结果在高维空间能够通过超平面分离，如图6.8所示。



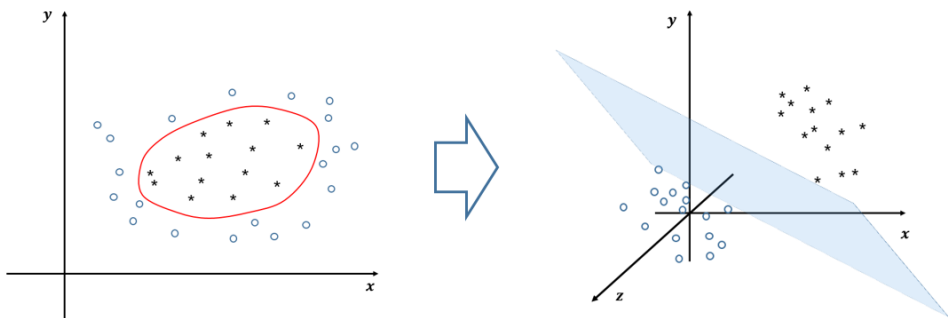


图6.8 利用核函数将二维线性不可分的数据映射为三维线性可分的数据

设原始输入特征向量 $\mathbf{x}$ 映射到高维空间后的向量为 $\phi(\mathbf{x})$ ，这样在高维空间可构建超平面 $\mathbf{w}^T \phi(\mathbf{x}) + \mathbf{b} = 0$ ，将 $\phi(\mathbf{x})$ 代替 $\mathbf{x}$ 可得到在高维空间的最优间隔分类器的最优化原始模型：

$$\begin{aligned} \max_{\mathbf{w}, \mathbf{b}} \quad & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{s. t.} \quad & \mathbf{y}^i \times (\mathbf{w}^T \phi(\mathbf{x}^i) + \mathbf{b}) \geq 1, i = 1, 2, \dots, m \end{aligned} \quad (6.49)$$

同理可得到其对偶问题为：

$$\begin{aligned} \min_a \quad & \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \mathbf{y}^i \mathbf{y}^j \mathbf{a}_i \mathbf{a}_j \left( \phi(\mathbf{x}^i) \right)^T \phi(\mathbf{x}^j) - \sum_{i=1}^m \mathbf{a}_i \\ \text{s. t.} \quad & \sum_{i=1}^m \mathbf{a}_i \mathbf{y}^i = 0, i = 1, 2, \dots, m \\ & \mathbf{a}_i \geq 0, i = 1, 2, \dots, m \end{aligned} \quad (6.50)$$

求解式(6.50)的关键是求解 $\left( \phi(\mathbf{x}^i) \right)^T \phi(\mathbf{x}^j)$ ，也就是特征映射后的内积，一般来说，被映射后的空间维度可能很高，甚至是无限维，并且 $\phi$ 的形式可能是未知的，因此，不直接求解内积，而是定义核函数，核函数接收两个原始空间的特征，用这个函数的结果来作为高维映射空间的向量内积结果，记为：

$$k(\mathbf{x}, \mathbf{y}) = \left( \phi(\mathbf{x}) \right)^T \phi(\mathbf{y}) \quad (6.51)$$

把式(6.51)代入式(6.50)，得到用核函数表示的最优化对偶模型：

$$\min_a \quad \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \mathbf{y}^i \mathbf{y}^j \mathbf{a}_i \mathbf{a}_j k(\mathbf{x}^i, \mathbf{x}^j) - \sum_{i=1}^m \mathbf{a}_i$$

$$\begin{aligned} \text{s.t. } \sum_{i=1}^m a_i y^i &= 0 \quad i = 1, 2, \dots, m \\ a_i &\geq 0 \quad i = 1, 2, \dots, m \end{aligned} \quad (6.52)$$

利用SMO算法求解出最优参数 $a^*$ 后，可得分类器的最终模型为：

$$\begin{aligned} f(x) &= w^T \phi(x) + b \\ &= \sum_{i=1}^m a_i y^i \left( \phi(x^i) \right)^T \phi(x) + b \\ &= \sum_{i=1}^m a_i y^i k \left( \phi(x^i), \phi(x) \right) + b \end{aligned} \quad (6.53)$$

考察式(6.53)与式(6.48)，可以发现没有添加核函数时，式(6.48)近似于单层感知机模型，而添加核函数后，相当于在神经网络中添加了隐藏层，如图6.9所示。

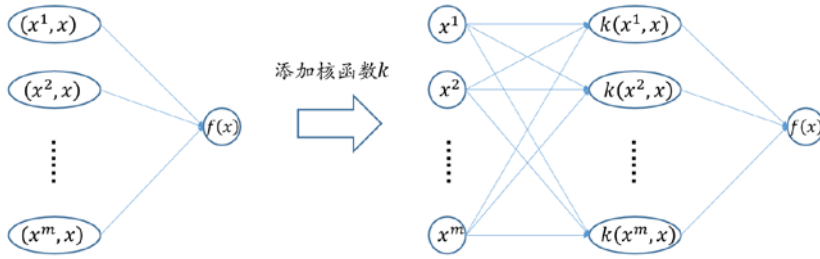


图6.9 从神经网络的角度来理解，添加核函数处理后，得到的模型比原始模型更强大

下面考察几个常用的核函数。

### 1. 多项式核

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + c)^d \quad (6.54)$$

为了讨论的方便，我们只考察多项式核的简单情形，不失一般性，令 $c = 0, d = 2$ ，即 $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y})^2$ ，设：

$$\mathbf{x} = (x_1, x_2, \dots, x_n)^T, \mathbf{y} = (y_1, y_2, \dots, y_n)^T$$

化简得：

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y})^2 = \left( \sum_{i=1}^n x_i y_i \right) \left( \sum_{i=1}^n x_i y_i \right) = \sum_{i=1}^n \sum_{j=1}^n (x_i x_j) (y_i y_j) \quad (6.55)$$

由式(6.55)的结果, 实际上得到映射函数 $\phi$ 的表达式为:

$$\phi(\mathbf{x}) = (x_1x_1, x_1x_2, \dots, x_1x_n, x_2x_1, \dots, x_2x_n, \dots, x_nx_1, \dots, x_nx_n)^T \quad (6.56)$$

也就是说, 经过 $\phi$ 的转换后, 数据从原来的 $n$ 维空间映射为 $n^2$ 维空间。

## 2. 高斯核

$$k(\mathbf{x}, \mathbf{y}) = \exp\left(\frac{-(\mathbf{x} - \mathbf{y})^2}{2\sigma^2}\right) \quad (6.57)$$

高斯核函数能够将原始的输入向量映射到无限维的向量空间中, 下面来看看其转换过程, 将式(6.57)展开, 得:

$$\begin{aligned} k(\mathbf{x}, \mathbf{y}) &= \exp\left(\frac{-(\mathbf{x} - \mathbf{y})^2}{2\sigma^2}\right) \\ &= \exp\left(\frac{-\mathbf{x}^T\mathbf{x} + 2\mathbf{x}^T\mathbf{y} - \mathbf{y}^T\mathbf{y}}{2\sigma^2}\right) \\ &= \exp\left(\frac{-\mathbf{x}^T\mathbf{x}}{2\sigma^2}\right) \exp\left(\frac{\mathbf{x}^T\mathbf{y}}{\sigma^2}\right) \exp\left(\frac{-\mathbf{y}^T\mathbf{y}}{2\sigma^2}\right) \end{aligned} \quad (6.58)$$

对指数 $\exp\left(\frac{\mathbf{x}^T\mathbf{y}}{\sigma^2}\right)$ , 利用泰勒展开得:

$$\exp\left(\frac{\mathbf{x}^T\mathbf{y}}{\sigma^2}\right) = \sum_{i=0}^{\infty} \frac{1}{i!} \left(\frac{\mathbf{x}^T\mathbf{y}}{\sigma^2}\right)^i = \sum_{i=0}^{\infty} \left[ \frac{1}{\sqrt{i!}} \left(\frac{\mathbf{x}^T}{\sigma}\right)^i \right] \times \left[ \frac{1}{\sqrt{i!}} \left(\frac{\mathbf{y}}{\sigma}\right)^i \right] = \varphi^T(\mathbf{x})\varphi(\mathbf{y}) \quad (6.59)$$

其中 $\varphi(\mathbf{x})$ 是无限维向量, 满足:

$$\varphi(\mathbf{x}) = (\varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}), \dots, \varphi_n(\mathbf{x}), \dots)^T, \varphi_i(\mathbf{x}) = \frac{1}{\sqrt{i!}} \left(\frac{\mathbf{y}}{\sigma}\right)^i \quad (6.60)$$

将式(6.60)代入式(6.58), 化简得:

$$\begin{aligned} k(\mathbf{x}, \mathbf{y}) &= \exp\left(\frac{-\mathbf{x}^T\mathbf{x}}{2\sigma^2}\right) \varphi^T(\mathbf{x})\varphi(\mathbf{y}) \exp\left(\frac{-\mathbf{y}^T\mathbf{y}}{2\sigma^2}\right) \\ &= \left[ \varphi(\mathbf{x}) \exp\left(\frac{-\mathbf{x}^T\mathbf{x}}{2\sigma^2}\right) \right]^T \left[ \varphi(\mathbf{y}) \exp\left(\frac{-\mathbf{y}^T\mathbf{y}}{2\sigma^2}\right) \right] \end{aligned}$$

令:

$$\phi(\mathbf{x}) = \varphi(\mathbf{x}) \exp\left(\frac{-\mathbf{x}^T\mathbf{x}}{2\sigma^2}\right) \quad (6.61)$$

可使得 $k(\mathbf{x}, \mathbf{y}) = [\phi(\mathbf{x})]^T[\phi(\mathbf{y})]$ 成立, 把式(6.60)代入式(6.61), 可得 $\phi(\mathbf{x})$ 的表达

形式，由于 $\phi(x)$ 是一个无限维向量，故 $\phi(x)$ 同样是一个无限维向量。

最后，怎么样的函数才可以成为核函数呢？一个核函数是合理的，只要它满足下面的定理。

**Mercer定理：**对于任意由 $m$ 个训练数据构成的数据集 $D$ ：

$$D = \{(x^1, y^1), (x^2, y^2), \dots, (x^m, y^m)\}$$

由核函数 $k(x, y)$ 作用于 $D$ 中任意的两个训练数据，得到的对称矩阵（核矩阵） $K$ 是半正定的，那么 $k(x, y)$ 是一个合理的核函数，其中核矩阵 $K$ 满足 $K_{ij} = k(x^i, x^j)$ ，如下所示：

$$K = \begin{bmatrix} k(x^1, x^1) & k(x^1, x^2) & \cdots & k(x^1, x^m) \\ k(x^2, x^1) & k(x^2, x^2) & \cdots & k(x^2, x^m) \\ \vdots & \vdots & \ddots & \vdots \\ k(x^m, x^1) & k(x^m, x^2) & \cdots & k(x^m, x^m) \end{bmatrix}$$

## 6.3 朴素贝叶斯

前面介绍了SVM和LR两种分类模型，它们都属于判别模型，由输入数据 $x$ 直接得到输出的估计值 $y = f(x)$ 或 $p(y|x)$ 。本节将介绍一个生成模型：朴素贝叶斯分类。

顾名思义，朴素贝叶斯分类是在贝叶斯分类的基础上做了一定的简化，首先来了解贝叶斯分类的相关知识。

给定输入数据 $x$ 的条件下，它属于类别 $c$ 的概率为 $p(c|x)$ ，前面讲解的两种模型都是直接得到 $p(c|x)$ 的结果，但根据贝叶斯定理，有：

$$p(c|x) = \frac{p(x|c) \times p(c)}{p(x)} \quad (6.62)$$

贝叶斯分类的思想是，不直接求取 $p(c|x)$ ，而是通过求取 $p(x|c)$ 和 $p(c)$ ，利用式(6.62)来找寻最优分类。注意，在一般情况下，不需要考虑 $p(x)$ 的值，因为，对于同一个输入数据， $p(x)$ 的值是相同的。

贝叶斯分类的难点在于求取 $p(x|c)$ ，则要求取在条件 $c$ 的条件下，输入属性值的联合分布，当输入属性很多时，这个计算量是非常大的，为此考虑这样一个假设，即输入数据的向量中，属性值是按类别条件独立，满足：

$$p(x|c) = p(x_1|c) \times p(x_2|c) \times \dots \times p(x_n|c) \quad (6.63)$$

朴素贝叶斯分类是一个概率模型，同样可以用概率图模型的观点来理解。事实上朴素贝叶斯网络是一种特殊的贝叶斯有向图模型，其模型可以用图6.10来表示。

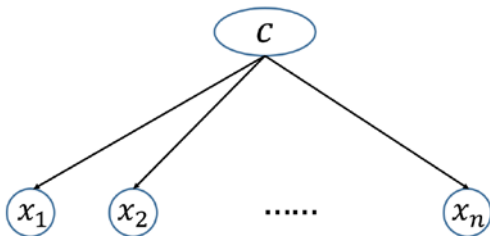


图6.10 朴素贝叶斯网络

由全局马尔科夫独立性可知，图6.10蕴含的全局独立性集合为：

$$I(G) = \{x_j \perp x_j | c: i = 1, 2, \dots, n, j = 1, 2, \dots, n \ i \neq j\}$$

要使用朴素贝叶斯模型进行分类，需要确定其参数，也就是求解 $p(c_k)$ 和 $p(x_i|c_k)$ 。参数训练的过程如下。

设由 $m$ 个训练数据构成的数据集 $D$ ：

$$D = \{(x^1, y^1), (x^2, y^2), \dots, (x^m, y^m)\}$$

其中：

$$x^i = (x_1^i, x_2^i, \dots, x_n^i)^T$$

$$y^i \in \{c_1, c_2, \dots, c_r\}$$

$$x_j^i \in \{a_1, a_2, \dots, a_p\}$$

$x^i$ 表示第 $i$ 个样本数据的输入， $y^i$ 表示第 $i$ 个样本数据的输出， $x_j^i$ 表示第 $i$ 个样本数据的输入特征向量的第 $j$ 个特征。参数的训练策略采用极大似然估计，也就是用训练数据的频率来估计概率：

$$p(c_k) = \frac{\sum_{i=1}^m 1_{y^i=c_k}}{m} \quad (6.64)$$

其中， $1_{y^i=c_k}$ 是一个指示函数，满足：

$$1_{y^i=c_k} = \begin{cases} 1 & y^i = c_k \\ 0 & y^i \neq c_k \end{cases} \quad (6.65)$$

同理，对于 $p(x_i|c_k)$ ，有：

$$p(x_j^i = a_t | y^i = c_k) = \frac{\sum_{i=1}^m 1_{y^i=c_k} x_j^i = a_t}{\sum_{i=1}^m 1_{y^i=c_k}}$$

(6.66)

最后需要注意的是，使用朴素贝叶斯网络进行分类时，需要注意零概率问题，由于训练数据是有限的，因此极有可能会出现 $p(x_i|c) = 0$ 的情形，这时候要用到平滑的策略来解决零概率问题。我们在12.5节讲解了几种常用的平滑技术，读者可以先阅读这一部分的相关知识，此外，读者也可以查阅文献[8]，文中对常用的平滑技巧有很好的总结。

6.4 树模型

决策树是机器学习中常用的一类算法模型，与前面的模型相比，它的优点是易于理解和实现，并且方便用可视化的方式展示。决策树是一种树模型结构，构建一棵决策树的算法流程如图6.11所示。

算法6.1 决策树分类算法

输入：训练数据集 $D$ ， $D = \{(x^1, y^1), (x^2, y^2), \dots, (x^m, y^m)\}$

其中输入特征向量 $x^i = \{x_1^i, x_2^i, \dots, x_n^i\}$

输出：生成一棵决策树 $T$

1. 初始化树节点node

2. 若数据集 $D$ 中的数据都属于同一类型 $C$ ，那么将节点node设为叶子节点，并且设置类别为 $C$ ，程序结束

3. 若数据集 $D$ 中剩下的数据中，特征向量的取值全部都相同，那么我们将节点node设为叶子节点，叶子节点的类别为 $D$ 中数量最多的类别，程序结束

4. 从当前的特征向量中挑选最好的特征向量 $x_j$

5. 对特征向量 $x_j$ 的每一个可能取值 $a$ ，执行下面的循环操作

5.1 为节点node生成一个子树 $T_s$

5.2 设 $D_s$ 为数据集 $D$ 中特征 $x_j$ 的取值为 $a$ 的所有子集

5.3 新的子数据集 $D_s$ 的特征向量为 $\{x_1, x_2, \dots, x_n\} - x_j$

5.4 if  $D_s$ 为空，则设置子树 $T_s$ 为叶节点，类别为 $D$ 中数量最多的类别

else 递归执行本函数，其中输入数据为子数据集 $D_s$ ，特征向量 $\{x_1, x_2, \dots, x_n\} - x_j$

生成子决策树 $T_s$

图6.11 构建决策树的算法

从图6.11可以看出，构建决策树是从根节点开始不断递归生成子树的过程。构建

162

决策树主要包括三个操作：特征选择、决策树的生成和决策树的剪枝。其中决策树的生成过程是一个递归的过程，本节主要讲解特征选择和剪枝的相关知识。

### 6.4.1 特征选择

从算法6.1的步骤4可知，构建决策树的过程中，在每一步，每一个非叶子节点都对应于一个特征，那么应该如何挑选当前最好的特征呢？挑选特征的方法有很多，不同的特征选取策略形成了不同决策树算法，但一个总的准则是：通过特征的选择，使得原来无序的数据逐渐变得有序，也就是使数据的纯度提升。本节介绍三种常用的选择指标：信息增益（information gain）、增益率和Gini指标。

#### 1. 信息增益

信息增益是ID3算法使用的特征选择指标。在第4章中，介绍了信息熵的概念，它是表示随机变量不确定性的度量，其数学表示为：

$$\text{info}(D) = - \sum_{i=1}^c p_i \times \log p_i \quad (6.67)$$

对于监督分类学习来说， $p_i$ 表示当前数据集 $D$ 中类别 $i$ 所占的比例。信息熵越小，表示数据集越稳定，纯度越高。假设按照特征 $x_j$ 来划分数数据集，设 $x_j$ 可以有 $v$ 个不同的取值，记为 $\{a_1, a_2, \dots, a_v\}$ ，利用 $x_j$ 的 $v$ 个不同的取值可以将数据集 $D$ 划分为 $v$ 个不同的子集，记为：

$$D = D_1 \cup D_2 \cup \dots \cup D_v \quad D_i \cap D_j = \emptyset, i \neq j$$

$D_i$ 表示在数据集 $D$ 中，特征 $x_j$ 取值为 $a_i$ 的全部子集，每一个子集用于生成对应的子树，在理想的情况下，我们希望通过特征 $x_j$ 的划分，能够将划分的不同子集数据都归为同一类别，也就是 $D_i$ 所包含的所有数据都属于同一类。信息熵是用来描述数据稳定性的度量，因此计算每一个子集 $D_i$ 的信息熵大小，并将它们相加，得到按特征 $x_j$ 划分数数据集后具有的信息量，记为：

$$\text{info}_{x_j}(D) = - \sum_{i=1}^v \frac{|D_i|}{|D|} H(D_i) \quad (6.68)$$

最后得到信息增益的计算公式为：

$$\text{Gain}(D, x_j) = \text{info}(D) - \text{info}_{x_j}(D) \quad (6.69)$$

信息增益越大，表示划分后的子数据集的纯度越高，因此，执行算法6.1的步骤4，计算每一个特征的信息增益，然后挑选信息增益最大的特征作为划分当前数据集的特征。

## 2. 增益率

信息增益的计算公式会偏向于取值较多的属性。从直观上来理解就是：当某一个特征的取值很多时，由它分裂产生的子树也越多，属于每一个子树的子数据集自然也比较少，因此，其纯度也会相对比较高。为了克服这个缺点，在信息增益的基础上产生了增益率。增益率也是C4.5算法使用的特征选择指标（注意，C4.5不是直接使用高增益率作为划分的特征，而是使用启发式的思想，先找出信息增益高于平均值的特征，然后从这些特征中挑选增益率最高的，更详细地分析读者可以参考文献[9]）。

增益率通过引入分裂信息（split information）来将信息增益规范化，对取值较多的特征起到一定的打压作用，分裂信息的定义为：

$$\text{split\_info}_{x_j}(D) = - \sum_{i=1}^v \frac{|D_i|}{|D|} \log \left( \frac{|D_i|}{|D|} \right) \quad (6.70)$$

增益率的定义为：

$$\text{Gain\_ratio}(D, x_j) = \frac{\text{Gain}(D, x_j)}{\text{split\_info}_{x_j}(D)} \quad (6.71)$$

执行算法6.1的步骤4时，计算每一个特征的增益率，然后挑选增益率最高的特征作为划分当前数据集的特征。

## 3. Gini指标

最后，讲解Gini指标（基尼指标），它是CART决策树采用的特征选择指标。首先定义数据集D的Gini值为：

$$\text{Gini}(D) = 1 - \sum_{i=1}^c (p_i)^2 \quad (6.72)$$

$p_i$ 表示当前数据集D中类别*i*所占的比例，与信息熵一样，Gini值同样被用来描述数据集D的纯度，Gini值越小，表示数据集D的纯度越高。进一步定义Gini指标：

$$\text{Gini\_index}(D, x_j) = \sum_{i=1}^v \frac{|D_i|}{|D|} \text{Gini}(D) \quad (6.73)$$



执行算法6.1的步骤4时，计算每一个特征的Gini指标，然后挑选Gini指标最小的特征作为划分当前数据集的特征。

## 6.4.2 剪枝策略

决策树的构建是一个递归过程，利用训练数据集从根结点开始，选取最合适的特征将数据集进行划分，因此决策树对训练数据的效果都很好，但也因此容易产生过拟合的现象。剪枝（pruning）是决策树解决过拟合问题的主要手段，通过剪枝操作可以消除不可靠的分支，同时也能够让整棵决策树变得更小、复杂度更低。

树剪枝的策略有预剪枝和后剪枝两种策略。

**预剪枝（prepruning）**是指在利用算法6.1构建决策树的过程中，通过提前停止程序的执行而实现的剪枝策略，一旦算法停止执行之后，该节点将成为叶子节点，一般会当前所包含的数据集中，数量最多的类别作为该叶子节点的类别。

那么应该如何确定是否需要提前终止呢？当求取了当前所有特征的某些选择指标（信息增益、增益率或Gini指标）后，我们需要评估分裂前后的优劣，具体的做法是：预先挑选一个阈值，如果当前的最优特征得到的指标都要比这个阈值低，那就可以提前终止算法的执行。

预剪枝是一种局部的贪心策略的思想，它根据当前的树结构，判别剪枝前后的效果对比。但有时可能会出现误剪枝的情况，即当前的划分可能效果不理想，但在这个划分基础上进行的后续划分可能会使整棵决策树的性能得到显著提升。此外，另一个先剪枝的缺点是阈值的选择是一个很困难的过程，阈值太大，会导致误剪枝的概率加大；阈值太小，将使得剪枝的优化效果不明显。

**后剪枝（postpruning）**是指在构建完决策树后，然后自底向上逐一考察每一个非叶子节点，比较合并前后的性能。如果以该非叶子节点为根的子树合并后的性能要更好，那么可以将该子树变为一个叶子节点，类别为当前子树所包含的数据集中数量最多的类别。

后剪枝因为是在构建完整棵树后才进行，因此它的风险更小，泛化能力要比预剪枝更好，但剪枝的速度要比预剪枝慢。

在预剪枝和后剪枝中，都需要比较分裂前后的效果，应该如何做呢？通常来说，会将整个训练数据划分为训练集和测试集，利用训练集构建决策树，然后利用测试集

检验合并子树前后的效果。

以后剪枝为例,为了讲解更加方便,假设特征1的取值为 $\{a_1, a_2, a_3\}$ ,类别为 $\{0,1\}$ ,训练数据与测试数据如图6.12所示。

数据集	特征1	类别
1	$a_1$	1
2	$a_1$	1
3	$a_2$	0
4	$a_3$	0
5	$a_1$	1
6	$a_2$	0
7	$a_1$	0
8	$a_3$	0

训练数据

数据集	特征1	类别
1	$a_1$	1
2	$a_1$	1
3	$a_2$	0
4	$a_3$	0

测试数据

图6.12 构建决策树的训练数据与测试数据

对特征1,如果树节点不分裂,由于训练数据集中,类别0的数目占多数,因此把节点的类别标记为类别0。然后将决策树作用于测试数据集,其准确率为50%;如果将节点分裂,则得到的决策树如图6.13所示,将该决策树应用于测试集合,得到的准确率为100%。这样就确信将特征1分裂后的效果要比不分裂的效果好。

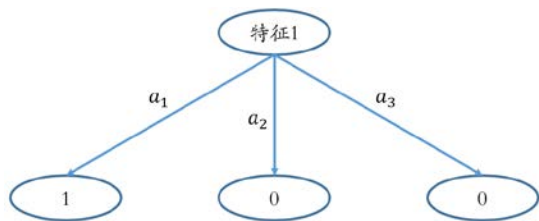


图6.13 将特征1分裂后的决策树

## 6.5 聚类

前面介绍了监督学习的常用算法模型,从本节开始,将会介绍无监督学习的相关知识。在无监督学习中,样本数据的标签信息是未知的,无监督学习的目标是从数据中挖掘出有用的特征和规律,并将这些特征和规律应用到后面的学习任务中。

聚类分析是最常见的无监督学习任务，具体来说，聚类分析是将一组数据按照内在的相似性划分为多个类别，使得同一个类别内的数据之间相似度较大，相互之间的距离较小；相反，对于不同类别的数据，它们之间的相似度较小，距离也比较远。首先给出聚类的形式化定义：设有由 $m$ 个训练数据构成的训练样本 $D$ ， $D$ 可表示为 $D = \{x^1, x^2, \dots, x^m\}$ ，其中 $x^i$ 表示第 $i$ 个训练样本数据，是一个 $n$ 维特征向量，记为 $x^i = \{x_1^i, x_2^i, \dots, x_n^i\}$ ，聚类的目标是将训练数据集 $D$ 划分为 $k$ 个互不相交的簇，记为 $\{C_1, C_2, \dots, C_k\}$ ，满足 $C_i \cap C_j = \emptyset$  ( $i \neq j$ )，且 $C_i \cup C_j = D$ 。

### 6.5.1 距离度量

聚类是一个基于距离划分数据集的过程，那么应该如何定义数据之间的距离呢？令 $d(x, y)$ 表示向量 $x$ 与 $y$ 之间的距离，则 $d(x, y)$ 需要满足下面的四个条件：

- 非负性： $d(x, y) \geq 0$ 。
- 同一性： $d(x, x) = 0$ 。
- 对称性： $d(x, y) = d(y, x)$ 。
- 三角不等式性： $d(x, z) \leq d(x, y) + d(y, z)$ 。

对于有序数据，常用的距离计算方法是Minkowski距离，假设当前有两个 $n$ 维空间的点 $x$ 和 $y$ ，满足：

$$x = (x_1, x_2, \dots, x_n)^T, y = (y_1, y_2, \dots, y_n)^T$$

则 $x$ 和 $y$ 的Minkowski距离的数学定义为：

$$d(x, y) = \left( \sum_{i=1}^m (x_i - y_i)^p \right)^{\frac{1}{p}} \quad (6.74)$$

当 $p = 2$ 时，就是常见的欧几里得距离；当 $p = 1$ ，则是曼哈顿距离。如果每一个特征的权重不一样，那么可以将式(6.74)修改为如下所示，其中 $w_i$ 是对应于第 $i$ 个特征的权重大小，修改后的距离称为加权Minkowski距离。

$$d(x, y) = \left( \sum_{i=1}^m w_i (x_i - y_i)^p \right)^{\frac{1}{p}} \quad (6.75)$$

此外，高斯距离也是一种常用的计算有序数据距离的方式。高斯距离的计算公式如式(6.76)所示，当 $x$ 与 $y$ 越接近时，它们的高斯距离越小；反之，当 $x$ 与 $y$ 越远时，高

斯距离就越大。

$$d(x,y)=\exp\left(-\frac{\|x-y\|}{2\sigma^2}\right) \tag{6.76}$$

对于无序数据，也就是某一个特征不是有序的数字时，比如特征 $x_i$ 可取的值为{西瓜,香蕉,雪梨}，那么不能直接使用Minkowski距离来衡量两者之间的距离，无序数据通常采用VDM距离来衡量，形式化定义为：设 $m_{ia}$ 表示在第 $i$ 个特征中取值为 $a$ 的样本数， $m_{iaj}$ 表示在第 $j$ 个簇中，第 $i$ 个特征取值为 $a$ 的样本数，设有 $k$ 个簇，则特征取值为 $a$ 和取值为 $b$ 的距离为：

$$\text{VDM}(a,b)=\sum_{j=1}^k\left(\frac{m_{iaj}}{m_{ia}}-\frac{m_{ibj}}{m_{ib}}\right)^p \tag{6.77}$$

当数据点中既含有有序特征，也含有无序特征时，将式(6.74)和式(6.77)合并，可得混合的距离计算公式：

$$d(x,y)=\left(\sum_{i=1}^s(x_i-y_i)^p+\sum_{i=s+1}^m\text{VDM}(x_i,y_i)\right)^{\frac{1}{p}} \tag{6.78}$$

6.5.2 层次聚类

层次聚类是将聚类看成是一个按层次进行数据划分的过程，算法结束后将形成一棵聚类树。层次聚类按照执行的顺序不同，可以分为自底向上的合并方法，以及自顶向下的分裂方法，如图6.14所示。

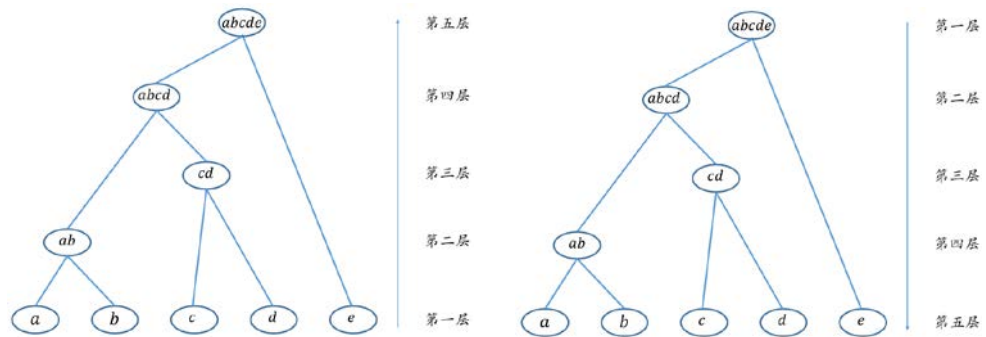


图6.14 左图，基于自底向上的层次聚类；右图，基于自顶向下的层次聚类

下面考察自底向上的AGNES算法的流程：初始时将每一个数据点单独作为一个簇，

然后选择距离最近的两个簇进行合并，进入下一层，反复进行这个操作，最终得到一个层次聚类树。整个流程的关键是如何比较两个簇的距离，如果两个簇就是两个点，那么它们的距离可以用6.5.2节提到的任意一个距离指标来度量，当两个簇中包含多个数据点时，衡量两个簇 $C_i$ 和 $C_j$ 的距离常用方法有3种，如图6.15所示。

- 最小距离：对于簇 $C_i$ 中的任意一个点 $p_i \in C_i$ ，以及簇 $C_j$ 中的任意一个点 $p_j \in C_j$ ，最小距离的定义如式(6.79)所示，最小距离由两个簇中最近的两个样本点决定。

$$d_{i,j} = \min d(p_i, p_j) \quad (6.79)$$

- 最大距离：对于簇 $C_i$ 中的任意一个点 $p_i \in C_i$ ，以及簇 $C_j$ 中的任意一个点 $p_j \in C_j$ ，最大距离的定义如式(6.80)所示，最大距离由两个簇中最远的两个样本点决定。

$$d_{i,j} = \max d(p_i, p_j) \quad (6.80)$$

- 平均距离：对于簇 $C_i$ 中的任意一个点 $p_i \in C_i$ ，以及簇 $C_j$ 中的任意一个点 $p_j \in C_j$ ，平均距离的定义如式(6.81)所示，平均距离由两个簇中所有的样本点决定。

$$d_{i,j} = \frac{1}{|C_i||C_j|} \sum_{p_i \in C_i} \sum_{p_j \in C_j} d(p_i, p_j) \quad (6.81)$$

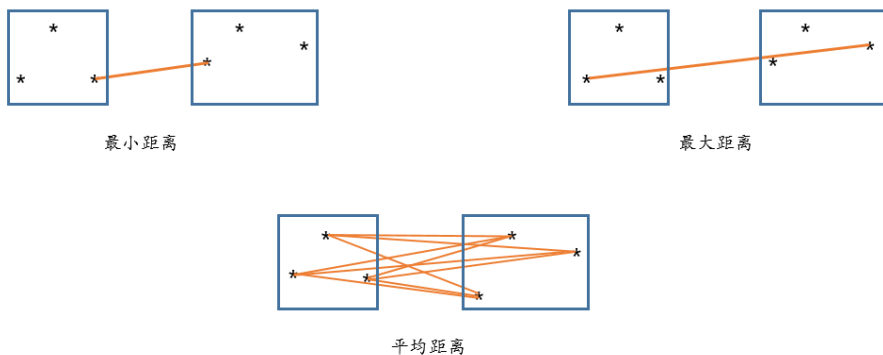


图6.15 三种常见的簇距离

AGNES算法流程如图6.16所示。

算法6.2    AGNES算法

- 输入：包含 $m$ 个数据点 $\{x^1, x^2, \dots, x^m\}$ 的数据集 $D$
- 输出： $k$ 个簇 $\{C_1, C_2, \dots, C_k\}$
1. 将每一个数据点单独初始化为一个簇，初始化 $C_i = x^i$ ，并记当前的簇数为 $cur = m$
  2. 当 $cur > k$ 时，执行下面的操作；否则跳出循环
    - 2.1 找出距离最短的两个簇 $C_i$ 和 $C_j(i < j)$ ，将簇 $C_i$ 和 $C_j$ 的数据点合并，合并后的簇重新标记为 $C_i$
    - 2.2 将簇 $C_{j+1}, C_{j+2}, \dots, C_{cur}$ 分别重新编号为 $C_j, C_{j+1}, \dots, C_{cur-1}$
    - 2.3 修改 $cur$ 值:  $cur = cur - 1$
  2. 输出 $k$ 个簇 $\{C_1, C_2, \dots, C_k\}$

图6.16    AGNES算法流程

自顶向下的DIANA算法流程，其过程与AGNES算法的过程相反，是一个逐层分裂的过程。初始时将全部数据点作为一个簇，然后按照某个准则将簇分裂，进入下一层，然后选择直径最大的簇，按照同样的准则进行分裂，如此反复进行这个操作，最终将所有的数据点分离。这里有一个新的定义就是簇的直径，簇的直径描述的是簇中最远的两个点的距离，如图6.17所示。

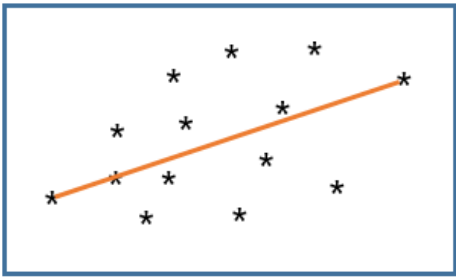


图6.17    簇的直径

DIANA算法流程如图6.18所示。

---

**算法6.3 DIANA算法**


---

输入：包含 $m$ 个数据点 $\{x^1, x^2, \dots, x^m\}$ 的数据集 $D$

输出： $k$ 个簇 $\{C_1, C_2, \dots, C_k\}$

1. 将全部数据点都初始化为一个簇，记为 $C_1 = \{x^1, x^2, \dots, x^m\}$ ，并记当前的簇数为 $cur = 1$
  2. 当 $cur < k$ 时，执行下面的操作；否则跳出循环
    - 2.1 找出直径最大的簇 $C_i$ 进行分裂操作
    - 2.2 找出簇中与其他点平均距离最大的点 $p$ ，将 $p$ 放入 $split\_group$ 中，其余点放在 $old\_part$ 中
    - 2.3 重复下面的操作
 

在 $old\_part$ 中找出到 $split\_group$ 最近的点的距离不大于到 $old\_part$ 中最近点的距离的点 $q$ ，将 $q$ 添加进 $split\_group$ 中；若没有找到这样的点则退出循环
    - 2.4 将 $split\_group$ 与 $old\_part$ 分别作为两个新的簇，编号分别为 $C_i$ 和 $C_{cur+1}$
    - 2.5  $cur = cur + 1$
- 

图6.18 DIANA算法流程

### 6.5.3 K-means聚类

K-means是当前最流行和经典的聚类方法之一，其核心思想是：对数据集 $D = \{x^1, x^2, \dots, x^m\}$ ，考虑所有可能的 $k$ 个簇集合，希望能找到一个簇集合 $\{C_1, C_2, \dots, C_k\}$ ，使得每一个点到其对应簇的中心的距离的平方和最小，即：

$$\min \left( \sum_{i=1}^k \sum_{x^j \in C_i} \|x^j - \mu_i\|^2 \right) \quad (6.82)$$

其中， $\mu_i = \frac{1}{|C_i|} \sum_{x^j \in C_i} x^j$ 表示簇 $C_i$ 的中心。但要找到满足式(6.82)的最小化条件的簇非常困难。K-means则采用了贪心的策略，通过迭代的方式来找到式(6.82)的近似解，算法首先随机挑选 $k$ 个样本点作为初始簇，对剩余的样本点，根据其到所有簇中心的距离，将点分配到最近的簇中，然后重新更新每一个簇的中心位置，不断重复这个过程，直到所有样本点不再变化为止，算法的代码流程如图6.19所示。

算法6.4	K-means算法
输入：包含 $m$ 个数据点 $\{x^1, x^2, \dots, x^m\}$ 的数据集 $D$	
输出： $k$ 个簇 $\{C_1, C_2, \dots, C_k\}$	
1. 若样本集的个数 $m \leq k$ ，则将每一个点单独作为一个簇，程序退出	
2. 任意从数据集 $D$ 中选取 $k$ 个点，作为初始的 $k$ 个簇	
3. 将剩余的数据点，分别计算到 $k$ 个簇的距离，把数据点分配到与其最近的簇中	
4. 重新计算每个簇的中心的位置	
5. 重复执行下面的操作，直到没有点需要被调整	
5.1 计算每一个点到簇的距离，并将数据点分配到与其最近的簇中	
5.2 重新计算每个簇的中心的位置	
6. 输出 $k$ 个簇 $\{C_1, C_2, \dots, C_k\}$	

图6.19 k-means算法流程

K-means聚类的时间复杂度为 $O(tnmk)$ ，其中 $t$ 表示迭代次数， $n$ 表示数据点个数， $m$ 表示特征数， $k$ 表示簇的数目，当数据量比较小的时候，K-means能够取得很好的效果。但当数据量比较大时，K-means的效率会下降，而且K-means比较容易陷入局部最优解。

6.5.4 谱聚类

谱聚类 (Spectral Clustering) 是一种基于图论的聚类方法，相比于前面的层次聚类和K均值聚类，谱聚类的效果要更好。谱聚类将数据点以图的形式展示，其中图的每一个顶点代表一个训练数据，图的边表示两个数据点之间的相似度，从而将聚类问题转化为图分割问题，每一个被分割的子图对应于一个簇，使得连接不同子图的边的权重尽可能低，同一个子图内的边的权重尽可能高。

将聚类转化为图算法问题，首先考虑如何合理构图，前面已经描述了构图的过程，最终得到的是一个完全图的结构，但完全图的边很多，为了使得算法的运行效率更高，可以去掉不必要的边，事实上，如果两个数据点的相似度非常低，那么它们在同一个簇中的概率也会非常小，基于这一思想，对每一个数据点 $x^i$ ，只考虑与其最相似的 $k$ 个点相连。构建后的图 $G$ 用其邻接矩阵来表示，也称为相似矩阵 $W$ ，为了讨论方便，下面给出关于图 $G$ 和相似矩阵 $W$ 的一些定义。

对于图 $G$ 中的两个子图 $A$ 和子图 $B$ ，定义 $A$ 子图与 $B$ 子图之间的所有边的权值之和如



下:

$$W(A, B) = \sum_{i \in A, j \in B} w_{ij} \quad (6.83)$$

其中,  $w_{ij}$  表示点  $i$  与点  $j$  的边权值, 也就是点  $i$  与点  $j$  的相似度。

对于子图  $A$ , 记  $\bar{A} = G - A$ , 表示子图  $A$  的补集。

如何通过图  $G$  来对数据点进行聚类呢? 由于同一个簇内的点相似度高, 而不同簇之间的点的相似度较低, 也就是相当于在图切割中, 那些被切断的边的权值之和最小, 而权重比较大的边没有被切断, 将聚类转化为图的最小割问题, 定义目标函数。

$$\min \text{cut}(A_1, A_2, \dots, A_k) = \min \left( \sum_{i=1}^k W(A_i, \bar{A}_i) \right) \quad (6.84)$$

我们的目标是最小化式(6.84), 但直接用最小割的算法, 往往效果不理想, 如图 6.20 所示, 以  $k = 2$  为例, 如果使用最小化式(6.84)的方法, 将得到割线  $A$ , 也就是得到两个簇, 分别为簇  $\{a\}$  和簇  $\{b, c, d, e, f\}$ , 但对于聚类来说, 这样的效果并不好, 单纯追求边权值的最小化, 将导致聚类数据可能出现不平衡的情况。如果采用割线  $B$ , 将得到的两个簇为  $\{a, b, c\}$  和  $\{d, e, f\}$ 。

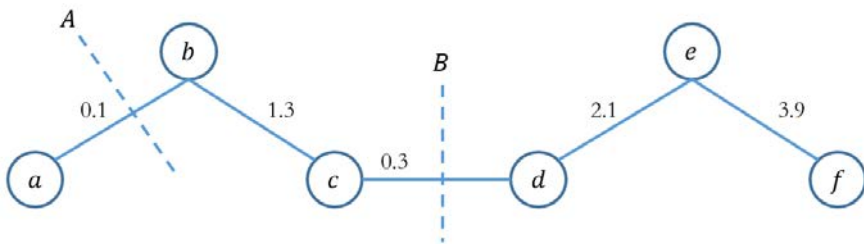


图6.20 图切割

为了使聚类的结果更加均衡, 需要修改式(6.84), 使得每个簇之间的点数尽量均衡, 因此, 在目标函数中加入每一个簇的数目, 修改后的目标函数变为如式(6.85)所示, 修改后的割也被称为RatioCut。

$$\min \text{RatioCut}(A_1, A_2, \dots, A_k) = \min \left( \sum_{i=1}^k \frac{W(A_i, \bar{A}_i)}{|A_i|} \right) = \min \left( \sum_{i=1}^k \frac{\text{cut}(A_i, \bar{A}_i)}{|A_i|} \right) \quad (6.85)$$

现在我们的目标是要最小化式(6.85), 但这个目标函数的最小化求解是非常困难的, 为此, 需要一种更加高效的方法。下面介绍拉普拉斯矩阵, 通过拉普拉斯矩阵来

高效求解式(6.85)<sup>[10]</sup>。

拉普拉斯矩阵的定义为： $\mathbf{L} = \mathbf{D} - \mathbf{W}$ ，其中 $\mathbf{W}$ 是前面提到的相似度矩阵， $\mathbf{D} = \{d_{ii}\}$ 表示度矩阵，它是一个对角矩阵，对角线的每一个元素 $d_{ii}$ 表示第 $i$ 个节点与其邻接节点的权重之和，也就是等于矩阵 $\mathbf{W}$ 的第 $i$ 行或者第 $i$ 列之和，即 $d_{ii} = \sum_{j=1}^m w_{ij}$ 。拉普拉斯矩阵有下面几个重要的性质。

- $\mathbf{L}$ 是对称的半正定矩阵。
- $\mathbf{L}$ 有 $m$ 个非负的特征值，则 $0 \leq \lambda_1, \lambda_2, \dots, \lambda_m$ ，其中 $m$ 是图 $G$ 的顶点个数。
- $\mathbf{L}$ 的最小特征值为0，且最小特征值对应的特征向量为 $\vec{\mathbf{1}} = (1, 1, \dots, 1)^T$ 。

设 $\mathbf{f}$ 是任意的 $m$ 维实向量，则满足：

$$\mathbf{f}^T \mathbf{L} \mathbf{f} = \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m w_{ij} (f_i - f_j)^2 \quad (6.86)$$

下面简要验证式(6.86)：

$$\begin{aligned} \mathbf{f}^T \mathbf{L} \mathbf{f} &= \mathbf{f}^T \mathbf{D} \mathbf{f} - \mathbf{f}^T \mathbf{W} \mathbf{f} = \sum_{i=1}^m d_i (f_i)^2 - \sum_{i=1}^m \sum_{j=1}^m f_i f_j w_{ij} \\ &= \frac{1}{2} \left( \sum_{i=1}^m d_i (f_i)^2 - 2 \sum_{i=1}^m \sum_{j=1}^m f_i f_j w_{ij} + \sum_{i=1}^m d_i (f_i)^2 \right) = \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m w_{ij} (f_i - f_j)^2 \end{aligned}$$

为了讨论方便，考虑二元聚类的情况，即 $k = 2$ 的情况，将数据点转化为图后，两个子图分别设为 $A$ 和 $\bar{A}$ ，令 $\mathbf{f}$ 满足：

$$f_i = \begin{cases} \sqrt{\frac{|\bar{A}|}{|A|}} & v_i \in A \\ -\sqrt{\frac{|A|}{|\bar{A}|}} & v_i \in \bar{A} \end{cases} \quad (6.87)$$

将式(6.87)代入式(6.86)可得：

$$\mathbf{f}^T \mathbf{L} \mathbf{f} = \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m w_{ij} (f_i - f_j)^2$$

$$\begin{aligned}
&= \frac{1}{2} \left( \sum_{i \in A, j \in \bar{A}} w_{ij} \left( \sqrt{\frac{|\bar{A}|}{|A|}} + \sqrt{\frac{|A|}{|\bar{A}|}} \right)^2 + \sum_{i \in A, j \in A} w_{ij} \left( -\sqrt{\frac{|\bar{A}|}{|A|}} - \sqrt{\frac{|A|}{|\bar{A}|}} \right)^2 \right) \\
&= \left( \sum_{i \in \bar{A}, j \in A} w_{ij} \right) \times \left( \frac{|\bar{A}|}{|A|} + \frac{|A|}{|\bar{A}|} + 2 \right) \\
&= \text{cut}(A, \bar{A}) \times \left( \frac{|\bar{A}| + |A|}{|A|} + \frac{|A| + |\bar{A}|}{|\bar{A}|} \right) = \text{RatioCut}(A, \bar{A}) \times m \quad (6.88)
\end{aligned}$$

由于 $m$ 是一个固定值,要最小化RatioCut,可转化为最小化 $\mathbf{f}^T \mathbf{L} \mathbf{f}$ ,且很容易推知 $\mathbf{f}$ 具有下面的性质。

- $\mathbf{f}^T \vec{1} = \sum_i f_i = 0 \Rightarrow \mathbf{f} \perp \vec{1}$ 。
- $\|\mathbf{f}\|^2 = \sum_i (f_i)^2 = m$ 。

这样得到新的最优化目标函数为:

$$\begin{aligned}
&\min_{\mathbf{f}} \mathbf{f}^T \mathbf{L} \mathbf{f} \\
&\text{s. t. } \mathbf{f} \perp \vec{1} \\
&\|\mathbf{f}\|^2 = m
\end{aligned} \quad (6.89)$$

其中,  $\mathbf{f} = \{f_i\}$ 满足式(6.87),  $m$ 为数据集 $D$ 的点的个数,即图 $G$ 的顶点个数,现在求解式(6.89)的最小值,由Rayleigh quotient理论可得<sup>[10]</sup>:

$$R(\mathbf{L}, \mathbf{f}) = \frac{\mathbf{f}^T \mathbf{L} \mathbf{f}}{\mathbf{f}^T \mathbf{f}} \quad (6.90)$$

其中,  $R$ 的最大值和最小值分别为 $L$ 的最大特征值和最小特征值,并且极值在 $\mathbf{f}$ 等于对应的特征向量时取得。由于 $\mathbf{f}^T \mathbf{f} = \|\mathbf{f}\|^2 = m$ 是一个常数,因此最小化 $\mathbf{f}^T \mathbf{L} \mathbf{f}$ 与最小化 $R(\mathbf{L}, \mathbf{f})$ 等价,但由于 $L$ 的最小特征值为0,其对应的特征向量为 $\vec{1}$ ,即 $\mathbf{f} = \vec{1}$ ,不满足 $\mathbf{f} \perp \vec{1}$ 的约束条件,因此转为求取第二小的特征值。这样很巧妙地解决了式(6.85)的NP难最小化问题。

考察式(6.89)的最小化结果如何转化为二元聚类中,当得到 $L$ 第二小的特征向量 $\mathbf{v}$ 时,  $\mathbf{v}$ 是一个 $m$ 维实数向量,但 $\mathbf{f}$ 的每一个元素 $f_i$ 只能取两个值(参见式(6.87)),那么应该如何将 $\mathbf{v}$ 转化为 $\mathbf{f}$ 呢?简单的方法是查看 $\mathbf{v}$ 的每一个元素 $v_i$ 是大于0还是小于0,如果大于0,则取值 $\sqrt{|\bar{A}|/|A|}$ ;如果小于0,则取值 $-\sqrt{|A|/|\bar{A}|}$ 。如果把求解得到的特征

向量 $\mathbf{v}$ 的每一个元素对应于一个点，那么根据每一个元素取值大于0还是小于0，可以将点划分为两类，这样我们将式(6.89)的最小化结果转化为聚类问题。

可以把其推广到 $k$ 元聚类中，具体的做法是求解拉普拉斯矩阵 $\mathbf{L}$ 的前 $k$ 小特征向量，构成一个大小为 $m \times k$ 的矩阵，每一行代表一个点，每一个点用 $k$ 维向量代替，对这个 $m$ 个 $k$ 维行向量执行 $K$ -means聚类，其结果就是原始的 $m$ 个点的聚类结果。

根据上面的分析，可以发现谱聚类的本质先对拉普拉斯矩阵进行降维，然后对降维后的矩阵执行  $K$ -means聚类得到。谱聚类算法流程如图6.21所示。

算法6.5    谱聚类算法流程
输入：包含 $m$ 个数据点 $\{x^1, x^2, \dots, x^m\}$ 的数据集 $D$
输出： $k$ 个簇 $\{C_1, C_2, \dots, C_k\}$
1. 由数据集 $D$ 构建图 $G$
2. 求取图 $G$ 的相似矩阵 $\mathbf{W}$ 和度矩阵 $\mathbf{D}$ ，并求得拉普拉斯矩阵 $\mathbf{L} = \mathbf{D} - \mathbf{W}$
3. 求取拉普拉斯矩阵的前 $k$ 个最小的特征向量，构成一个大小为 $m \times k$ 的矩阵 $\mathbf{M}$
4. 对矩阵 $\mathbf{M}$ 的行向量按 $K$ -means方法进行聚类，得到 $k$ 个簇 $\{C_1, C_2, \dots, C_k\}$
5. 矩阵 $\mathbf{M}$ 的第 $i$ 行对应所属的簇就是原数据点 $x^i$ 对应的簇

图6.21 谱聚类算法流程

参考文献：

[1] 周志华. 机器学习. 清华大学出版社; 2016 年. ISBN: 7302423288, 9787302423287.

[2] 李航. 统计学习方法. 清华大学出版社; 2012 年. ISBN: 9787302275954, 7302275955.

[3] Kevin P. Murphy. Machine Learning, A Probabilistic Perspective. The MIT Press. 2012.

[4] Andrew Ng, John Duchi. CS229: Machine Learning.

[5] McCullagh, Peter; Nelder, John (1989). Generalized Linear Models, Second Edition. Boca Raton: Chapman and Hall/CRC. ISBN 0-412-31760-5.

[6] Cortes, C.; Vapnik, V. (1995). "Support-vector networks". Machine Learning. 20 (3): 273–297.

[7] John Platt. Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. 1998.

[8] Stanley F. Chen, Joshua Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. Computer Speech and Language (1999) 13, 359–394.

[9] Quinlan, J. R. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.

[10] U Von Luxburg. A tutorial on spectral clustering. Statistics and computing, 2007.

# 7

## 数值计算与最优化

数值计算与最优化理论是上世纪中期形成和发展起来的一门应用数学分支学科，广泛应用于机器学习、工程设计、生产管理、交通运输、国防等重要领域。而其中机器学习又是与最优化理论联系最紧密的学科之一，机器学习的模型训练，最终都归结为最优化问题，也就是寻找最优的参数，使得模型的误差损失函数最小，而寻找最优参数的方法称为最优化方法。本章将深入分析在深度学习领域常用的最优化方法及其原理，并分析不同的优化方法之间各自的特点。

由于求解 $\max(f(x))$ 的最大化问题可以等价地转化为求解最小值 $\min(-f(x))$ ，为了讨论方便，不失一般性，本章仅讨论求解最小值的最优化问题。

### 7.1 无约束极小值的最优化条件

假设现有 $n$ 元函数 $f(x_1, x_2, \dots, x_n)$ ，若在点 $x^0 = (x_1^0, x_2^0, \dots, x_n^0)^T$ 附近存在邻域 $\delta(x^0)$ ，使得对于任意的 $x \in \delta(x^0)$ ，都有 $f(x) \geq f(x^0)$ 成立，那么称点 $x^0$ 是函数 $f(x)$ 的极小值点，当 $f(x)$ 为凸函数时，点 $x^0$ 也是函数 $f(x)$ 的最小值点。

若点 $x^0 = (x_1^0, x_2^0, \dots, x_n^0)^T$ 为 $f(x)$ 的局部极小值点，那么显然点 $x_i^0$ 也为一元函数 $g(x_i) = f(x_1^0, x_2^0, \dots, x_{i-1}^0, x_i, x_{i+1}^0, \dots, x_n^0)$ 的局部极小值点，由一元函数极值的必要条件可知：

$$\frac{\partial g(x_i)}{\partial x_i} \Big|_{x_i=x_i^0} = \frac{\partial f(x_1^0, x_2^0, \dots, x_{i-1}^0, x_i, x_{i+1}^0, \dots, x_n^0)}{\partial x_i} \Big|_{x_i=x_i^0} = 0 \quad (7.1)$$

由此，可以得到下面有关多元函数极小值点的必要条件。

**定理7.1:** 设函数 $f(x)$  ( $x \in \mathbf{R}^n$ ) 在点 $x^0 = (x_1^0, x_2^0, \dots, x_n^0)^T$  处可微，且 $x^0$  为 $f(x)$  的局部极小值点，则必有梯度 $\nabla f(x^0) = 0$  成立。

满足条件 $\nabla f(x^0) = 0$  的点也称为 $f(x)$  的驻点或稳定点，定理7.1给出了判断多元函数极小值点存在的必要条件，但要注意的是，该定理反之并不成立，即满足梯度 $\nabla f(x^0) = 0$  的点不一定是 $f(x)$  的局部极小值点，有关多元函数极值点的充分条件，可以参考相关文献[1]，这里不再阐述。

利用极值的必要条件，把求解函数极小值问题，转化为求解下面的方程组问题：

$$\nabla f(x) = 0$$

即求解点 $x = (x_1, x_2, \dots, x_n)$ ，满足：

$$\left\{ \begin{array}{l} \frac{\partial f}{\partial x_1} = 0 \\ \frac{\partial f}{\partial x_2} = 0 \\ \vdots \\ \frac{\partial f}{\partial x_n} = 0 \end{array} \right. \quad (7.2)$$

式(7.2)是一个包含 $n$ 个未知数， $n$ 个方程的方程组，对于一些特殊的函数，比如目标函数 $f(x)$  为二次函数时，式(7.2)是一个线性方程组，利用解析法（如高斯消元法、矩阵三角分解法等）可以较方便求解，但遗憾的是， $f(x)$  一般是很复杂的非线性函数，求解非线性方程组并不比求解最优化问题简单，甚至比求解最优化问题更困难、更复杂。因此，在机器学习领域，求解最优化问题，一般不会通过解析法来求解式(7.2)，而是通过数值计算方法来直接求取函数极值。

迭代法是数值计算最常用的最优化方法，它的基本思想是：首先给定 $f(x)$  的一个极小值点的初始估计 $x^0$ ，然后通过迭代的方式得到点序列 $\{x^t\} (t = 1, 2, \dots)$ ，如果这个

点序列的极限 $x^*$ 逼近极小值点,那么称这个序列 $\{x^t\}(t = 0, 1, \dots)$ 为极小化序列,因此,最优化问题转化为应该如何得到这个极小化的点序列?

也就是说,假设已经得到 $(x^0, x^1, \dots, x^k)$ ,那么通过什么方式来得到新的点 $x^{k+1}$ 呢?在欧几里得空间中,点 $x^{k+1}$ 与点 $x^k$ 之差是一个向量,一个向量由其方向和长度来决定,因此迭代公式可以写成:

$$x^{k+1} = x^k + \lambda_k \mathbf{d}_k \quad (7.3)$$

其中 $\mathbf{d}_k$ 是一个方向向量, $\lambda_k$ 称为步长(或学习率),当 $\lambda_k$ 和 $\mathbf{d}_k$ 都被确定后,也就可以唯一确定点 $x^{k+1}$ ,并依此迭代,最后求得极小值点。

从上面的分析可知,迭代法的关键是确定步长 $\lambda_k$ 和方向向量 $\mathbf{d}_k$ ,事实上,后面我们也会看到,各种迭代算法的区别就在于得到步长 $\lambda_k$ 和方向 $\mathbf{d}_k$ 的方式不同。

选择 $\lambda_k$ 和 $\mathbf{d}_k$ 的方法有很多,也因此产生不同的迭代算法,通常,一个好的迭代算法应满足以下两个条件。

第一,递减性。通过式(7.3)构造的极小化序列应该是逐步下降的,也就是说,构造的点序列 $\{x^k\}$ ,对应的函数值序列 $\{f(x^k)\}$ 应满足:

$$f(x^0) \geq f(x^1) \geq \dots \geq f(x^k) \geq \dots$$

第二,收敛性。收敛性是迭代算法设计必不可少的条件。点序列必须能收敛到极小值点,否则,构造这个序列就失去意义了。

## 7.2 梯度下降

梯度下降是神经网络最常用的优化方法之一,因此,在很多优秀的深度学习工具库中都内置了梯度下降算法接口。

梯度下降的迭代方向 $\mathbf{d}$ 由函数 $f$ 的一阶导数(梯度)决定,这也是梯度下降算法的名字由来,首先来回顾一下梯度的几何意义。

假设现有 $n$ 元函数 $f(x_1, \dots, x_n)$ ,函数在某点 $x^0 = (x_1^0, x_2^0, \dots, x_n^0)^T$ 处的梯度是一个向量:

$$\mathbf{d} = \left( \frac{\partial f}{\partial x_1} \Big|_{x_1=x_1^0}, \frac{\partial f}{\partial x_2} \Big|_{x_2=x_2^0}, \dots, \frac{\partial f}{\partial x_n} \Big|_{x_n=x_n^0} \right)^T$$

它的方向就是 $f$ 在该点 $x^0$ 处函数值增长最快的方向。基于梯度的这个性质，如果把迭代的每一步沿着当前点的梯度方向的反方向进行迭代，那么就能得到一个逐步递减的极小化序列。

根据每一次迭代所使用的训练数据集范围的不同，可以把梯度下降算法区分为：批量梯度下降、随机梯度下降和小批量梯度下降。

## 1. 批量梯度下降

批量梯度下降（Batch Gradient Descent，简称BGD），也称之为最速下降法，误差损失函数由全量训练数据的误差构成，因此，当数据量很大的时候，速度会非常慢，同时，它不能以在线的方式更新模型，也就是说，当训练数据有新元素加入时，需要对全量的数据进行更新，效率很低，因此，当前的梯度下降法一般都不会采用BGD策略。

## 2. 随机梯度下降

随机梯度下降（Stochastic Gradient Descent，简称SGD），SGD策略是对BGD的改进，SGD每一次更新，只考虑一个样本数据的误差损失，因此，它的速度要远远优于BGD。更主要的是，它能进行在线的参数更新。但是SGD也有一些缺点，包括：由于单个样本会出现相似或重复的情况，因此，数据的更新会出现冗余；此外，单个数据之间的差异会比较大会比较大，造成每一次迭代的损失函数会出现较大的波动，如图7.1所示。

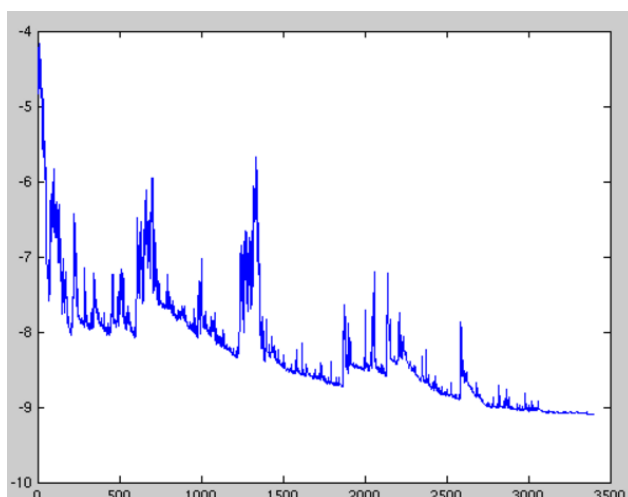


图7.1 SGD在运算过程中，损失函数会出现较大的波动（图片摘自维基百科）



### 3. 小批量梯度下降

小批量梯度下降( Mini-Batch Gradient Descent, 简称 MBGD ), MBGD结合了BGD和SGD的优点, 并克服了BGD和SGD存在的缺点。MBGD的更新策略是指每次的参数更新, 优化的目标函数是由 $n$ 个样本数据构成,  $n$ 的值一般较小, 一般取10到500之间, 这种做法有3个优点。

- 每一批的数据量较小, 特别适合高效的矩阵运算, 尤其是GPU的并行加速, 因此虽然MBGD的训练数据要比SGD多, 但效率上与SGD差别不大。
- 与SGD相比, MBGD每一批考虑了更多的样本数据, 每一批数据之间的整体差异更小、更平均, 结果也更稳定。因此不会出现像图7.1那样的频繁波动情况。
- 由于效率与SGD相当, 因此MBGD策略同样适用于在线的模型更新。

一般来说, 当前的梯度下降算法普遍采用MBGD策略, 因此, 在后面的讲解中, 除非有特别的说明, 否则我们所指的梯度下降都是指小批量的梯度下降, 即MBGD。第7.1节提到, 对于无约束最优化问题的求解, 关键在于确定步长 $\lambda_k$ 和方向向量 $\mathbf{d}_k$ , 下面考察梯度下降中用于确定步长 $\lambda_k$ 和方向向量 $\mathbf{d}_k$ 的几个不同算法策略:

#### 7.2.1 传统更新策略

传统的更新策略, 也被称之为**vanilla策略**, 是梯度下降算法中最简单的参数更新策略, 参数沿着其梯度的反方向变化, 假设参数向量是 $\boldsymbol{\theta}$ , 梯度为 $d\boldsymbol{\theta}$ , 则参数的更新策略为:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - lr \times d\boldsymbol{\theta} \quad (7.4)$$

其中,  $lr$ 是学习率, 它是一个预先设置的固定值超参数。

优点: 算法简单高效, 当 $lr$ 的值较小的时候, 算法能够保证得到全局最优解(凸函数)或局部最优解(非凸函数)。

缺点: 传统的迭代策略主要存在两个缺点。

(1) 算法的效率受到 $lr$ 的影响较大,  $lr$ 太小, 会导致算法收敛速度过慢;  $lr$ 过大, 容易在迭代过程中出现震荡现象, 甚至无法收敛, 如图7.2所示。



图7.2  $lr$ 过大，导致出现发散的情况

(2) 在较为平坦的平面区域, 由于梯度接近于0, 因此每一次迭代的变化非常小, 造成训练效率下降, 甚至会被误判为最优解而提前终止, 如图7.3所示。



图7.3 在平坦地区，迭代速度非常慢

传统梯度的更新策略的伪代码如下所示, 其中,  $lr$ 为学习率,  $param$ 是模型参数向量,  $d(param)$ 是指对当前参数向量 $param$ 求取梯度。

```
>>> for epoch in range(n_epoch):  
    v = lr * d(param)  
    param = param - v
```

对于传统的梯度下降算法, 参数点序列 $\{\theta^0, \theta^1, \dots\}$ 的生成与迭代方向向量 $d_k$ 的关系如图7.4所示:

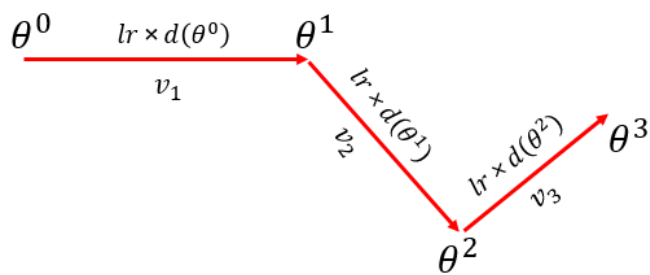


图7.4 传统梯度下降的方向变化图

### 7.2.2 动量更新策略

回顾传统的梯度更新策略，我们发现每一次迭代的方向 $d_k$ 等于当前batch数据集的误差损失函数的梯度，但是，很多时候不同batch之间数据变化比较大，这样就会造成迭代方向来回震动的情况，从而减缓了收敛的速度，甚至导致发散。

动量（momentum，有些文献也称之为冲量）更新策略<sup>[2]</sup>，它从物理学的角度出发，来解决传统的梯度更新策略中存在的问题。一个物体在运动时具有惯性，把这个思想运用到梯度下降中，即更新时在一定程度上保留之前更新的方向的同时，也利用当前batch的梯度微调最终的更新方向，因此，整个更新策略既保留了稳定性，也能根据实际的数据变化做出相应的调整。

具体来说，每一次迭代更新，迭代方向 $d_k$ 都由两部分构成。

第一部分是上一时刻的迭代方向，即动量或惯性，记为 $\mu \times v_{i-1}$ ， $v_{i-1}$ 是上一时刻的迭代方向， $\mu$ 是动量系数。

第二部分是当前batch样本集合的梯度方向，记为 $lr \times d(\theta^{i-1})$ 。

$$v_i = \mu \times v_{i-1} + lr \times d(\theta^{i-1}) \quad (7.5)$$

$$\theta^i = \theta^{i-1} - v_i \quad (7.6)$$

如图7.5所示，每一次的迭代方向等于上一时刻迭代方向与当前的梯度方向的向量之和。

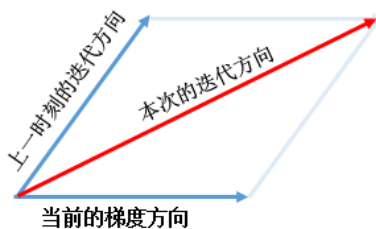


图7.5 动量更新策略的迭代方向

当上一次的迭代方向与当前batch的梯度方向相反时，momentum策略能起到一个减速的作用，防止走偏；相反，当上一次的迭代方向与当前的batch梯度方向相同时，momentum能起到一个加速的作用，避免在平坦的曲面上迭代过慢。

momentum的更新步骤如下面的伪代码所示，其中 $\mu$ 是惯性系数， $lr$ 是学习率，

$d(\text{param})$ 是当前batch的梯度。

```
>>> for epoch in range(n_epoch):
    v = mu*v + lr*d(param)
    param = param - v
```

参数点序列 $\{\theta^0, \theta^1, \dots\}$ 的生成与迭代方向向量 $\mathbf{d}_k$ 的变化关系如图7.6所示：

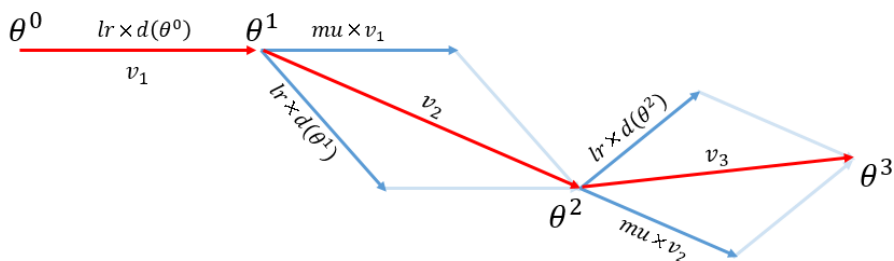


图7.6 动量更新策略的方向变化图

### 7.2.3 改进的动量更新策略

改进的动量更新策略（Nesterov Accelerated Gradient，简称NAG），也被称为Nesterov Momentum。首先来观察上一节的动量更新策略，动量更新策略仍然存在可以优化的地方，观察式(7.5)，可以把迭代的方向分为两步走，第一步是沿着上次的更新方向继续前进 $\text{mu} \times v_{i-1}$ ，到达点 $(\theta^{i-1})'$ ，即有：

$$(\theta^{i-1})' = \theta^{i-1} - \text{mu} \times v_{i-1}$$

然后沿着点 $\theta^{i-1}$ 的梯度方向前进 $\text{lr} \times d(\theta^{i-1})$ ，到达点 $\theta^i$ ，故有：

$$\theta^i = (\theta^{i-1})' - \text{lr} \times d(\theta^{i-1})$$

但仔细观察，我们发现第二步的当前点已经不是 $\theta^{i-1}$ ，而是新的点 $(\theta^{i-1})' = \theta^{i-1} - \text{mu} \times v_{i-1}$ ，那么一个很自然的想法是可否在第二步改为求取点 $(\theta^{i-1})'$ 的梯度呢？改进的动量更新策略的方向变化，如图7.7所示。

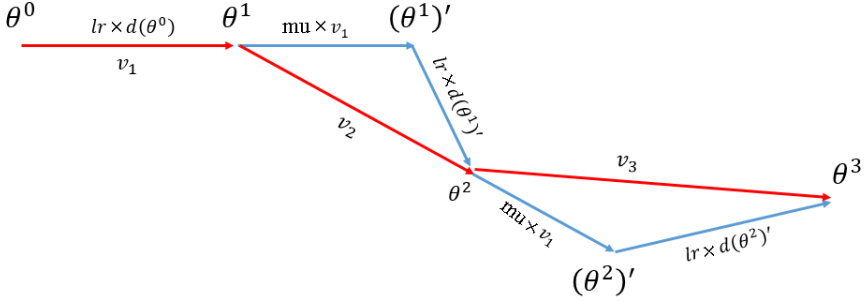


图7.7 改进的动量更新策略的方向变化图

把前面的文字表述用公式化来表示，可以得到NAG策略的初始计算形态：

$$(\theta^{i-1})' = \theta^{i-1} - \mu \times v_{i-1} \quad (7.7)$$

$$v_i = \mu \times v_{i-1} + lr \times d((\theta^{i-1})') \quad (7.8)$$

$$\theta^i = \theta^{i-1} - v_i \quad (7.9)$$

这种做法被称为“向前看”策略，这个细微的改进会带来哪些好处呢？通过下面详细推导来深入分析，首先来化简 $\theta^i - \mu \times v_i$ ：

$$\begin{aligned} \theta^i - \mu \times v_i &= \theta^{i-1} - v_i - \mu \times v_i = \theta^{i-1} - (\mu + 1) \times v_i \\ &= \theta^{i-1} - (\mu + 1) \times (\mu \times v_{i-1} + lr \times d(\theta^{i-1} - \mu \times v_{i-1})) \\ &= \theta^{i-1} - \mu \times v_{i-1} - \mu^2 \times v_{i-1} - lr \times (\mu + 1) \times d(\theta^{i-1} - \mu \times v_{i-1}) \end{aligned} \quad (7.10)$$

令：

$$\hat{\theta}^i = \theta^i - \mu \times v_i \quad (7.11)$$

$$\widehat{\theta^{i-1}} = \theta^{i-1} - \mu \times v_{i-1} \quad (7.12)$$

$$\begin{aligned} \hat{v}_i &= \mu^2 \times v_{i-1} + lr \times (\mu + 1) \times d(\theta^{i-1} - \mu \times v_{i-1}) \\ &= \mu^2 \times v_{i-1} + lr \times (\mu + 1) \times d(\widehat{\theta^{i-1}}) \end{aligned} \quad (7.13)$$

把式(7.11)、式(7.12)、式(7.13)分别代入式(7.10)，可得：

$$\hat{\theta}^i = \widehat{\theta^{i-1}} - \hat{v}_i \quad (7.14)$$

同时，继续对式(7.13)进行展开：

$$\begin{aligned}
\widehat{v}_i &= \mu^2 \times v_{i-1} + lr \times (\mu + 1) \times d(\theta^{i-1} - \mu \times v_{i-1}) \\
&= \mu^2 \times v_{i-1} + lr \times (\mu + 1) \times d(\widehat{\theta^{i-1}}) \\
&= lr \times (\mu + 1) \times d(\widehat{\theta^{i-1}}) + \mu^2 \times v_{i-1} \\
&= lr \times (\mu + 1) \times d(\widehat{\theta^{i-1}}) + \mu^2 \times (\mu \times v_{i-2} + lr \times d(\theta^{i-2} - \mu \times v_{i-2})) \\
&= lr \times (\mu + 1) \times d(\widehat{\theta^{i-1}}) + \mu^2 \times (\mu \times v_{i-2} + lr \times d(\widehat{\theta^{i-2}})) \\
&= lr \times (\mu + 1) \times d(\widehat{\theta^{i-1}}) + \mu^2 \times lr \times d(\widehat{\theta^{i-2}}) + \mu^3 \times v_{i-2} \\
&= lr \times (\mu + 1) \times d(\widehat{\theta^{i-1}}) + \mu^2 \times lr \times d(\widehat{\theta^{i-2}}) + \mu^3 \times lr \times d(\widehat{\theta^{i-3}}) + \mu^4 \times v_{i-3} \\
&= lr \times (\mu + 1) \times d(\widehat{\theta^{i-1}}) + \mu^2 \times lr \times d(\widehat{\theta^{i-2}}) + \mu^3 \times lr \times d(\widehat{\theta^{i-3}}) + \\
&\quad \mu^4 \times lr \times d(\widehat{\theta^{i-4}}) + \mu^5 \times v_{i-4} \\
&= lr \times (\mu + 1) \times d(\widehat{\theta^{i-1}}) + \mu^2 \times lr \times d(\widehat{\theta^{i-2}}) + \mu^3 \times lr \times d(\widehat{\theta^{i-3}}) + \\
&\quad \mu^4 \times lr \times d(\widehat{\theta^{i-4}}) + \mu^5 \times lr \times d(\widehat{\theta^{i-5}}) + \dots
\end{aligned} \tag{7.15}$$

把式(7.15)的结果代入  $\mu \times \widehat{v_{i-1}}$ ，化简得：

$$\begin{aligned}
\mu \times \widehat{v_{i-1}} &= \mu \times lr \times (\mu + 1) \times d(\widehat{\theta^{i-2}}) + \mu^3 \times lr \times d(\widehat{\theta^{i-3}}) + \\
&\quad \mu^4 \times lr \times d(\widehat{\theta^{i-4}}) + \mu^5 \times lr \times d(\widehat{\theta^{i-5}}) + \dots
\end{aligned} \tag{7.16}$$

把式(7.16)的结果代入  $\widehat{v}_i - \mu \times \widehat{v_{i-1}}$ ，化简得：

$$\widehat{v}_i - \mu \times \widehat{v_{i-1}} = lr \times d(\widehat{\theta^{i-1}}) + \mu \times lr \times (d(\widehat{\theta^{i-1}}) - d(\widehat{\theta^{i-2}})) \tag{7.17}$$

$$\Leftrightarrow \widehat{v}_i = \mu \times \widehat{v_{i-1}} + lr \times d(\widehat{\theta^{i-1}}) + \mu \times lr \times (d(\widehat{\theta^{i-1}}) - d(\widehat{\theta^{i-2}})) \tag{7.18}$$

联合式(7.14)和式(7.18)，得到NAG原始形式（式(7.7)、式(7.8)、式(7.9)）的另一种替代表示形式：

$$\widehat{v}_i = \mu \times \widehat{v_{i-1}} + lr \times d(\widehat{\theta^{i-1}}) + \mu \times lr \times (d(\widehat{\theta^{i-1}}) - d(\widehat{\theta^{i-2}})) \tag{7.19}$$

$$\widehat{\theta^i} = \widehat{\theta^{i-1}} - \widehat{v}_i \tag{7.20}$$

观察式(7.19)与式(7.5)的区别，我们发现NAG策略比momentum策略增加了最后的一项  $\mu \times lr \times (d(\widehat{\theta^{i-1}}) - d(\widehat{\theta^{i-2}}))$ ，而我们知道：

$$d(\widehat{\theta^{i-1}}) - d(\widehat{\theta^{i-2}}) \approx \nabla_{\theta}^2 f \tag{7.21}$$

也就是说最后的一项是目标函数的二阶导数的近似，由于利用了二阶导数的信息，因此从理论上来说，NAG比momentum的效率要更高。

### 7.2.4 自适应梯度策略

前面介绍了几种不同的梯度下降策略，但仔细观察，我们发现前面的策略都是针对迭代的方向进行优化，而步长是固定的，即所有参数共享相同的学习率，学习率在每一个方向上的大小固定，这样很容易造成算法被卡在鞍点的位置（*saddle point*）。

如图7.8所示，当迭代下降到a点处，由于学习率固定不变，容易在鞍点处来回震荡，无法找到最优点。本节考察学习率随迭代次数变化而变化的自适应梯度策略（adaptive gradient algorithm，简称 AdaGrad），AdaGrad是一种自适应的调整学习率的方法，首先给出AdaGrad的每一次迭代的公式：

$$acc_i = acc_{i-1} + (d(\theta_{i-1}))^2 \quad (7.22)$$

$$\theta_i = \theta_{i-1} - \frac{lr}{\sqrt{acc_i + \epsilon}} \times d(\theta_{i-1}) \quad (7.23)$$

其中， $lr$ 是全局学习率， $\epsilon$ 是一个非常小的常数，目的是为了防止分母为0。观察式(7.23)，迭代的方向由梯度方向 $d(\theta_{i-1})$ 来决定，但学习率 $lr$ （或步长）不再是固定值。下面分析AdaGrad是如何有效解决鞍点问题的。

（1）观察式(7.22)，当迭代时间越长，则累加的梯度（ $acc$ ）越大，使得式(7.23)的学习率会随着时间减少，在接近目标函数的最小值时，不会因为学习率过大而走过头。

（2）不同的参数之间学习率不相同，因此，与前面的固定学习率算法相比，不容易卡在马鞍点。

（3）继续观察式(7.22)与式(7.23)，可以发现，如果参数梯度累加比较小，则学习率会比较大，所以参数迭代的步长会比较大；相反，如果参数梯度累加比较大，则学习率会比较小，参数迭代的步长会比较小，因此，AdaGrad特别适合于稀疏参数网络的最优化求解，因为对于稀疏权重参数，不太重要的参数（0值或接近于0值的参数），其导数累加之和会比较小，我们可以放心加大步伐前进，相反，对于较为重要的参数，导数累加之和会比较大，对于这些参数的迭代步长需要谨慎。

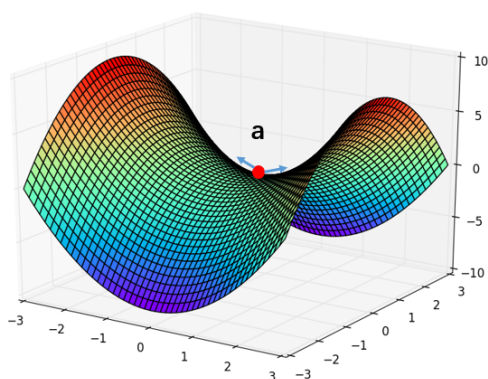


图7.8 马鞍点

除了上面提到的四种常见的梯度下降策略外，当前常用的梯度下降策略还包括Adadelta、RMSprop、Adam等，它们都是在前面的基础上进一步对学习率进行了优化，特别是Adadelta和RMSprop，它们在递归神经网络中的表现要远远优于SGD，限于本书的篇幅，我们不再展开具体的讨论，更详细的细节，读者可以参考文献[8]。

## 7.3 共轭梯度

首先考察求解下列二次型的无约束最优解问题的最优化方法，其中 $\mathbf{Q}$ 是实对称矩阵。

$$f(x) = \frac{1}{2}x^T \mathbf{Q}x + b^T x + c, x = (x_1, x_2, \dots, x_n)^T \quad (7.24)$$

如果可以将函数 $f(x)$ 化为变量分离函数，即：

$$f(x) = f_1(x_1) + f_2(x_2) + \dots + f_n(x_n)$$

那么只要从任意的一点 $x^0$ 出发，分别沿每一个坐标轴方向进行一维搜索，共进行 $n$ 次迭代后，一定能得到 $f(x)$ 的最优解。

而对于形如式(7.24)的二次函数，如果能找到一组正交基 $P = \{p_1, p_2, \dots, p_n\}$ ，使得 $p_i$ 满足：

$$p_i^T \mathbf{Q} p_j = 0, i \neq j \quad (7.25)$$

则称 $p_i$ 与 $p_j$ 是 $\mathbf{Q}$ 共轭的。在新的基下， $f(x)$ 将变为变量分离函数，具体来说，这是因为：



$$\begin{aligned}
f(Px) &= \frac{1}{2}(Px)^T Q(Px) + b^T(Px) + c \\
&= \sum_{i=1}^n \left( \frac{1}{2} x_i^T p_i^T Q p_i x_i + b^T(p_i x_i) \right) + c
\end{aligned} \tag{7.26}$$

其中, 设  $f_i(x_i) = \frac{1}{2} x_i^T p_i^T Q p_i x_i + b^T(p_i x_i)$ , 则式(7.26)可化简为:

$$f(Px) = f_1(x_1) + f_2(x_2) + \cdots + f_n(x_n) + c \tag{7.27}$$

化简为变量分离函数后, 就可以在每一个基向量方向上进行一维搜索。因此, 二次函数式(7.24)的最优化问题归结为寻找有效的基向量  $p_1, p_2, \dots, p_n$ , 使得对于当  $i \neq j$  时, 有  $p_i^T Q p_j = 0$  成立。

对式(7.24)求导, 有:

$$\nabla f(x) = Qx + b \tag{7.28}$$

于是对于任意给定的  $x, y \in \mathbf{R}^n$ , 有:

$$\nabla f(x) - \nabla f(y) = Q(x - y) \tag{7.29}$$

任意选取初始点  $x^1$ , 初始方向与梯度下降一样,  $p_1 = -\nabla f(x_1)$ , 假设现在已经得到  $(x^2, p_2), (x^3, p_3), \dots, (x^k, p_k)$ , 且  $p_1, p_2, \dots, p_k$  两两  $Q$  共轭, 令:

$$x^{k+1} = x^k + a_k p_k \tag{7.30}$$

其中,  $a_k$  的值由  $a_k = \operatorname{argmin}(f(x^k + a_k p_k))$  求得, 故有:

$$\frac{df(x^k + a_k p_k)}{da} \Big|_{a=a_k} = p_k^T \times \nabla f(x^{k+1}) = 0$$

$$\Leftrightarrow p_k^T \times (Q(x^{k+1} - x^k) + \nabla f(x^k)) = 0$$

$$\Leftrightarrow p_k^T \times (Q a_k p_k + \nabla f(x^k)) = 0$$

化简得  $a_k = \frac{-p_k^T \nabla f(x^k)}{p_k^T Q p_k}$ 。

此外, 对于任意的  $j \leq k$ , 利用最小值的必要条件, 同样有:

$$\frac{df(x^j + a_j p_j)}{da} \Big|_{a=a_j} = p_j^T \times \nabla f(x^{j+1}) = 0 \tag{7.31}$$

成立, 事实上, 更一般的, 我们马上可以证明  $p_j^T \times \nabla f(x^{k+1}) = 0$ ,  $j = 1, 2, \dots, k$  都成立, 对于任意给定的  $j \leq k$ , 由式(7.29)有:

$$\begin{aligned}
\nabla f(x^{k+1}) - \nabla f(x^{j+1}) &= \mathbf{Q}(x^{k+1} - x^{j+1}) \\
\Leftrightarrow p_j^T \times \nabla f(x^{k+1}) &= p_j^T \times \nabla f(x^{j+1}) + p_j^T \times \mathbf{Q}(x^{k+1} - x^{j+1}) \\
\Leftrightarrow p_j^T \times \nabla f(x^{k+1}) &= p_j^T \times \mathbf{Q}(x^{k+1} - x^{j+1}) \\
\Leftrightarrow p_j^T \times \nabla f(x^{k+1}) &= p_j^T \times \mathbf{Q}[(x^{k+1} - x^k) + (x^k - x^{k-1}) + \cdots + (x^{j+2} - x^{j+1})] \\
\Leftrightarrow p_j^T \times \nabla f(x^{k+1}) &= p_j^T \times \mathbf{Q}[a_k p_k + a_{k-1} p_{k-1} + \cdots + a_{j+1} p_{j+1}] \\
\Leftrightarrow p_j^T \times \nabla f(x^{k+1}) &= p_j^T \times \mathbf{Q} \left[ \sum_{i=j+1}^k a_i p_i \right] \\
\Leftrightarrow p_j^T \times \nabla f(x^{k+1}) &= \left[ \sum_{i=j+1}^k a_i (p_j^T \times \mathbf{Q} \times p_i) \right] \\
\Leftrightarrow p_j^T \times \nabla f(x^{k+1}) &= 0
\end{aligned} \tag{7.32}$$

令：

$$p_{k+1} = -\nabla f(x^{k+1}) + \lambda_k p_k \tag{7.33}$$

其中 $\lambda_k$ 由：

$$p_k^T \mathbf{Q} p_{k+1} = p_k^T \mathbf{Q} (-\nabla f(x^{k+1})) + p_k^T \mathbf{Q} (\lambda_k p_k) = 0 \tag{7.34}$$

确定，得：

$$\lambda_k = \frac{p_k^T \mathbf{Q} \nabla f(x^{k+1})}{p_k^T \mathbf{Q} p_k} \tag{7.35}$$

由式(7.30)和式(7.33)，确定了下一次迭代的点坐标和方向 $(x^{k+1}, p_{k+1})$ 。最后要证明的是，对于任意的 $j = 1, 2, \dots, k-1$ ，均有 $p_j^T \mathbf{Q} p_{k+1}$ 成立（其中 $p_k^T \mathbf{Q} p_{k+1} = 0$ 已经由式(7.34)作为前提条件给出）。

事实上，因为：

$$p_j^T \mathbf{Q} p_{k+1} = p_j^T \mathbf{Q} (-\nabla f(x^{k+1})) + p_j^T \mathbf{Q} (\lambda_k p_k) \tag{7.36}$$

由于 $j < k$ ，且 $p_1, p_2, \dots, p_k$ 两两 $\mathbf{Q}$ 共轭，故上式的最后一项为0，进一步化简：

$$\begin{aligned}
p_j^T \mathbf{Q} p_{k+1} &= p_j^T \mathbf{Q} (-\nabla f(x^{k+1})) \\
\Leftrightarrow a_j p_j^T \mathbf{Q} p_{k+1} &= a_j p_j^T \mathbf{Q} (-\nabla f(x^{k+1})) = \left( a_j p_j^T \mathbf{Q} (-\nabla f(x^{k+1})) \right)^T
\end{aligned}$$

$$\begin{aligned}
&\Leftrightarrow a_j p_j^T Q p_{k+1} = (-\nabla f(x^{k+1}))^T Q (a_j p_j) \\
&\Leftrightarrow a_j p_j^T Q p_{k+1} = (-\nabla f(x^{k+1}))^T Q (x^{j+1} - x^j) \\
&\Leftrightarrow a_j p_j^T Q p_{k+1} = (-\nabla f(x^{k+1}))^T (\nabla f(x^{j+1}) - \nabla f(x^j)) \quad (7.37)
\end{aligned}$$

由式(7.32)和式(7.33), 对  $j < k$ , 有:

$$(-\nabla f(x^{k+1}))^T \nabla f(x^j) = (\lambda_k p_k^T - p_{k+1}^T) \nabla f(x^j) = 0 \quad (7.38)$$

把式(7.38)代入式(7.37)有:

$$\begin{aligned}
&a_j p_j^T Q p_{k+1} = (-\nabla f(x^{k+1}))^T (\nabla f(x^{j+1}) - \nabla f(x^j)) \\
&\Leftrightarrow a_j p_j^T Q p_{k+1} = (-\nabla f(x^{k+1}))^T \nabla f(x^{j+1}) - (-\nabla f(x^{k+1}))^T \nabla f(x^j) \\
&\Leftrightarrow a_j p_j^T Q p_{k+1} = 0 \quad (7.39)
\end{aligned}$$

式(7.34)和式(7.39)证明了对于  $j \leq k$ ,  $p_{k+1}$  与  $p_j$  是两两  $Q$  共轭。

图7.9给出共轭梯度法的算法步骤。

---

#### 算法7.1 共轭梯度法步骤

---

求解  $\min f(x)$ , 其中  $f(x) = \frac{1}{2} x^T Q x + b^T x + c$   $x = (x_1, x_2, \dots, x_n)^T$

1. 任意选择初始点  $x^1 \in \mathbf{R}^n$ , 令  $p_1 = \nabla f(x^1)$
2. 若  $\nabla f(x^k) = 0$ , 则算法停止; 否则  $x^{k+1} = x^k + a_k p_k$

$$\text{其中 } a_k = \frac{-p_k^T \nabla f(x^k)}{p_k^T Q p_k}$$

$$p_{k+1} = -\nabla f(x^{k+1}) + \lambda_k p_k$$

$$\lambda_k = \frac{p_k^T Q \nabla f(x^{k+1})}{p_k^T Q p_k}$$

3.  $k = k + 1$ , 返回2
- 

图7.9 二次型函数的共轭梯度最优化算法步骤

对于非二次型方程  $f(x)$ , 可以通过在某一点  $x^k$  进行泰勒展开, 将其转化为二次函数, 即:

$$f(x) \approx f(x^k) + \nabla f(x^k) \times (x - x^k) + \frac{1}{2} (x - x^k)^T \times \nabla^2 f(x^k) \times (x - x^k) \quad (7.40)$$

其中,  $\mathbf{Q} = \nabla^2 f(x^k)$  为函数  $f(x)$  在点  $x^k$  处的**海森矩阵**, 但对于  $n \times n$  维的海森矩阵, 直接套用算法 7.1 计算量会很大, 为此, 首先将  $\lambda_k$  化成没有  $\mathbf{Q}$  的形式:

$$\lambda_k = \frac{p_k^T \mathbf{Q} \nabla f(x^{k+1})}{p_k^T \mathbf{Q} p_k} = \frac{a_k^T p_k^T \mathbf{Q} \nabla f(x^{k+1})}{a_k^T p_k^T \mathbf{Q} p_k} \quad (7.41)$$

由于海森矩阵是一个对称矩阵, 则  $\mathbf{Q}^T = \mathbf{Q}$ , 上式可进一步化简为:

$$\begin{aligned} \lambda_k &= \frac{p_k^T \mathbf{Q} \nabla f(x^{k+1})}{p_k^T \mathbf{Q} p_k} = \frac{a_k^T p_k^T \mathbf{Q} \nabla f(x^{k+1})}{a_k^T p_k^T \mathbf{Q} p_k} = \frac{(\mathbf{Q} p_k a_k)^T \times \nabla f(x^{k+1})}{(\mathbf{Q} p_k a_k)^T p_k} \\ &= \frac{(\mathbf{Q}(x^{k+1} - x^k))^T \times \nabla f(x^{k+1})}{(\mathbf{Q}(x^{k+1} - x^k))^T \times p_k} = \frac{(\nabla f(x^{k+1}) - \nabla f(x^k))^T \times \nabla f(x^{k+1})}{(\nabla f(x^{k+1}) - \nabla f(x^k))^T \times p_k} \\ &\Leftrightarrow \lambda_k = \frac{\|\nabla f(x^{k+1})\|^2}{\|\nabla f(x^k)\|^2} \end{aligned} \quad (7.42)$$

这样就求出对于任意二阶可导的函数  $f(x)$ , 其共轭梯度的求解算法, 如图 7.10 所示。

---

#### 算法 7.2 共轭梯度法步骤

---

求解  $\min f(x)$ , 其中  $f(x)$  为二阶可导的任意函数,  $x = (x_1, x_2, \dots, x_n)^T$

1. 任意选择初始点  $x^1 \in R^n$ , 令  $p_1 = \nabla f(x^1)$
2. 若  $\nabla f(x^k) = 0$ , 则算法停止; 否则  $x^{k+1} = x^k + a_k p_k$

其中  $a_k = \operatorname{argmin} (f(x^k + a_k p_k))$

$$p_{k+1} = -\nabla f(x^{k+1}) + \lambda_k p_k$$

$$\lambda_k = \frac{\|\nabla f(x^{k+1})\|^2}{\|\nabla f(x^k)\|^2}$$

3.  $k = k + 1$ , 返回 2
- 

图 7.10 任意的函数, 对应共轭梯度最优优化算法步骤

## 7.4 牛顿法

首先, 对于任意函数  $f(x)$ , 利用泰勒展开, 将任意一个二阶可微的函数  $f(x)$  转化为下面的二次型近似表示:

$$f(x) \approx f(x^k) + \nabla f(x^k) \times (x - x^k) + \frac{1}{2}(x - x^k)^T \times \nabla^2 f(x^k) \times (x - x^k)$$

设 $x^*$ 是函数 $f(x)$ 的极值, 由极值的必要条件可得:

$$\begin{aligned} \nabla f(x^*) &= \nabla f(x^k) + \nabla^2 f(x^k) \times (x^* - x^k) = 0 \\ \Leftrightarrow x^* &= x^k - \frac{\nabla f(x^k)}{\nabla^2 f(x^k)} \end{aligned} \quad (7.43)$$

这样可以从任意初始点 $x^0$ 开始, 第 $k$ 步的迭代方向为 $d_k = \nabla f(x^k)$ , 步长为 $\lambda_k = (\nabla^2 f(x^k))^{-1}$ , 构造出下面的牛顿迭代公式:

$$x^{k+1} = x^k - \frac{\nabla f(x^k)}{\nabla^2 f(x^k)} \quad (7.44)$$

与梯度下降法相比, 牛顿迭代的迭代方向 $d_k$ 仍然为 $\nabla f(x^k)$ , 但其步长 $\lambda_k$ 由海森矩阵的逆 $(\nabla^2 f(x^k))^{-1}$ 来决定。具体来说, 梯度下降只考虑在当前点的一阶导数信息, 即梯度 $(\nabla f(x^k))$ ; 而牛顿法不仅考虑当前点的梯度 $(\nabla f(x^k))$ , 还考虑到了二阶信息, 即走这一个方向后, 坡度是否会变得更大 $(\nabla^2 f(x^k))$ , 这与NAG有异曲同工之妙, 但牛顿法不是对二阶导数的模拟, 而是真正考察了函数的二阶性质。

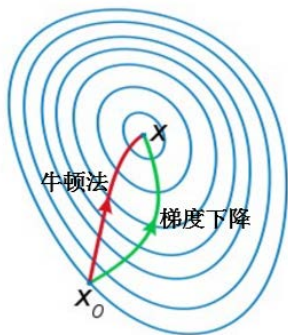


图7.11 牛顿法与梯度下降法的路径选择对比

通常把 $d_k = (\nabla^2 f(x^k))^{-1} \times \nabla f(x^k)$ 作为一个整体看待, 称为牛顿方向。牛顿法虽然具有全局收敛性和速度快等优点, 但它存在最大的问题是, 每一次迭代都需要计算函数 $f$ 的海森矩阵及其逆矩阵。当 $n$ 值很大时, 这个计算量不但非常巨大, 而且存储量也很惊人, 因此牛顿法更多的是作为理论上的参考, 在实际的应用中, 通常采用拟牛顿法来求解。

## 7.5 拟牛顿法

前面提到，牛顿法在计算函数 $f$ 的海森矩阵及其逆矩阵时的代价非常巨大，拟牛顿法的核心思想是如何更准确地模拟海森矩阵（或其逆矩阵）。形式化来说，就是用一个矩阵 $\mathbf{H}_k$ 来近似第 $k$ 次迭代时的海森矩阵逆矩阵 $(\nabla^2 f(\mathbf{x}^k))^{-1}$ ，从而在第 $k$ 次迭代时的迭代公式变为：

$$\begin{aligned} d_k &= \mathbf{H}_k \nabla f(\mathbf{x}^k) \\ \mathbf{x}^{k+1} &= \mathbf{x}^k - a_k d_k \end{aligned} \quad (7.45)$$

各种拟牛顿法的区别就在于近似海森矩阵逆矩阵时，所采用的策略不一样。下面首先考察拟牛顿条件，然后给出3个常用的拟牛顿法及其原理分析。

### 7.5.1 拟牛顿条件

由公式(7.45)可以得到经过 $k+1$ 次迭代后，到达点 $\mathbf{x}^{k+1}$ ，在点 $\mathbf{x}^{k+1}$ 对 $f(\mathbf{x})$ 进行泰勒展开，得：

$$f(\mathbf{x}) \approx f(\mathbf{x}^{k+1}) + \nabla f(\mathbf{x}^{k+1}) \times (\mathbf{x} - \mathbf{x}^{k+1}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{k+1})^T \times \nabla^2 f(\mathbf{x}^{k+1}) \times (\mathbf{x} - \mathbf{x}^{k+1})$$

对上式两边进行求导，有：

$$\begin{aligned} \nabla f(\mathbf{x}) &= \nabla f(\mathbf{x}^{k+1}) + \nabla^2 f(\mathbf{x}^{k+1}) \times (\mathbf{x} - \mathbf{x}^{k+1}) \\ \Leftrightarrow \nabla f(\mathbf{x}) - \nabla f(\mathbf{x}^{k+1}) &= \nabla^2 f(\mathbf{x}^{k+1}) \times (\mathbf{x} - \mathbf{x}^{k+1}) \end{aligned} \quad (7.46)$$

取 $\mathbf{x} = \mathbf{x}^k$ ，且令 $\mathbf{g}_k = \nabla f(\mathbf{x}^k)$ ，整理得：

$$\begin{aligned} \nabla f(\mathbf{x}^{k+1}) - \nabla f(\mathbf{x}^k) &= \nabla^2 f(\mathbf{x}^{k+1}) \times (\mathbf{x}^{k+1} - \mathbf{x}^k) \\ \Leftrightarrow \mathbf{g}_{k+1} - \mathbf{g}_k &= \nabla^2 f(\mathbf{x}^{k+1}) \times (\mathbf{x}^{k+1} - \mathbf{x}^k) \end{aligned} \quad (7.47)$$

令：

$$\begin{aligned} \mathbf{s}_k &= \mathbf{x}^{k+1} - \mathbf{x}^k, \\ \mathbf{y}_k &= \mathbf{g}_{k+1} - \mathbf{g}_k, \\ \mathbf{H}_{k+1} &\approx (\nabla^2 f(\mathbf{x}^{k+1}))^{-1}, \end{aligned}$$

$$\mathbf{B}_{k+1} = (\mathbf{H}_{k+1})^{-1} \approx \nabla^2 f(\mathbf{x}^{k+1})$$

则式(7.47)可进一步化简为:

$$\mathbf{y}_k \approx \mathbf{B}_{k+1} \times \mathbf{s}_k \quad (7.48)$$

或

$$\mathbf{s}_k \approx \mathbf{H}_{k+1} \times \mathbf{y}_k \quad (7.49)$$

我们把式(7.48)或式(7.49)称为拟牛顿条件, 也称为 $secant$ 等式, 它定义了 $\mathbf{B}_k$ 对海森矩阵 $\nabla^2 f(\mathbf{x}^k)$ 进行模拟约束, 以及 $\mathbf{H}_k$ 对海森矩阵的逆 $(\nabla^2 f(\mathbf{x}^k))^{-1}$ 进行模拟约束。

### 7.5.2 DFP算法

DFP算法由Davidon在1959年提出, 经Fletcher和Powell在1963年改进而形成。DFP算法的核心思想是求解海森矩阵逆 $(\nabla^2 f(\mathbf{x}^k))^{-1}$ 的近似矩阵 $\mathbf{H}_k$ , 它定义了下面的迭代公式来求解 $\mathbf{H}_k$ :

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \mathbf{E}_k \quad (7.50)$$

其中, 初始值 $\mathbf{H}_0 = \mathbf{I}$ ,  $\mathbf{E}_k$ 称为校正矩阵, 它的表示形式为:

$$\mathbf{E}_k = a_k \mathbf{U}_k \mathbf{U}_k^T + b_k \mathbf{V}_k \mathbf{V}_k^T \quad (7.51)$$

把式(7.51)和式(7.50)代入式(7.49), 可得:

$$\begin{aligned} \mathbf{s}_k &\approx (\mathbf{H}_k + a_k \mathbf{U}_k \mathbf{U}_k^T + b_k \mathbf{V}_k \mathbf{V}_k^T) \times \mathbf{y}_k \\ \Leftrightarrow \mathbf{s}_k - \mathbf{y}_k \mathbf{H}_k &\approx a_k \mathbf{U}_k \mathbf{U}_k^T \mathbf{y}_k + b_k \mathbf{V}_k \mathbf{V}_k^T \mathbf{y}_k \end{aligned} \quad (7.52)$$

令:

$$\begin{aligned} \mathbf{s}_k &= a_k \mathbf{U}_k \mathbf{U}_k^T \mathbf{y}_k \\ -\mathbf{y}_k \mathbf{H}_k &= b_k \mathbf{V}_k \mathbf{V}_k^T \mathbf{y}_k \end{aligned}$$

取:

$$\mathbf{U}_k = \mathbf{s}_k, \quad a_k = \frac{1}{\mathbf{U}_k^T \mathbf{y}_k} = \frac{1}{\mathbf{s}_k^T \mathbf{y}_k} \quad (7.53)$$

$$\mathbf{V}_k = -\mathbf{y}_k \mathbf{H}_k, \quad b_k = \frac{1}{\mathbf{V}_k^T \mathbf{y}_k} = \frac{-1}{\mathbf{y}_k^T \mathbf{H}_k \mathbf{y}_k} \quad (7.54)$$

将式(7.53)和式(7.54)代入式(7.50)，得到 $\mathbf{H}_k$ 的迭代公式为：

$$\mathbf{H}_{k+1} = \mathbf{H}_k + \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{s}_k^T \mathbf{y}_k} - \frac{\mathbf{H}_k \mathbf{y}_k \mathbf{y}_k^T \mathbf{H}_k}{\mathbf{y}_k^T \mathbf{H}_k \mathbf{y}_k} \quad (7.55)$$

式(7.55)就是对海森矩阵逆矩阵的直接近似，把式(7.55)代入式(7.45)便可以得到完整的DFP迭代算法。

### 7.5.3 BFGS算法

BFGS算法是由Broyden、Fletcher、Goldfarb和Shanno四位研究人员在1970年发明的一种拟牛顿法。它的思想与DFP算法很相似，BFGS算法的核心思想是求解海森矩阵 $\nabla^2 f(\mathbf{x}^k)$ 的模拟矩阵 $\mathbf{B}_k$ ，它定义了下面的迭代公式来求解 $\mathbf{B}_k$ 。

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \mathbf{E}_k \quad (7.56)$$

其中，初始值 $\mathbf{B}_0 = \mathbf{I}$ ， $\mathbf{E}_k$ 称为校正矩阵，它的表示形式与前面DFP校正矩阵一致：

$$\mathbf{E}_k = a_k \mathbf{U}_k \mathbf{U}_k^T + b_k \mathbf{V}_k \mathbf{V}_k^T \quad (7.57)$$

把式(7.57)和式(7.56)代入式(7.48)，可得：

$$\begin{aligned} \mathbf{y}_k &\approx (\mathbf{B}_k + a_k \mathbf{U}_k \mathbf{U}_k^T + b_k \mathbf{V}_k \mathbf{V}_k^T) * \mathbf{s}_k \\ \Leftrightarrow \mathbf{y}_k - \mathbf{s}_k \mathbf{B}_k &\approx a_k \mathbf{U}_k \mathbf{U}_k^T \mathbf{s}_k + b_k \mathbf{V}_k \mathbf{V}_k^T \mathbf{s}_k \end{aligned} \quad (7.58)$$

令：

$$\begin{aligned} \mathbf{y}_k &= a_k \mathbf{U}_k \mathbf{U}_k^T \mathbf{s}_k \\ -\mathbf{s}_k \mathbf{B}_k &= b_k \mathbf{V}_k \mathbf{V}_k^T \mathbf{s}_k \end{aligned}$$

取：

$$\mathbf{U}_k = \mathbf{y}_k, \quad a_k = \frac{1}{\mathbf{U}_k^T \mathbf{s}_k} = \frac{1}{\mathbf{y}_k^T \mathbf{s}_k} \quad (7.59)$$

$$\mathbf{V}_k = -\mathbf{s}_k \mathbf{B}_k, \quad b_k = \frac{1}{\mathbf{V}_k^T \mathbf{s}_k} = \frac{-1}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k} \quad (7.60)$$

将式(7.60)和式(7.59)代入式(7.56)，得到 $\mathbf{B}_k$ 的迭代公式为：

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^T \mathbf{B}_k}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k} \quad (7.61)$$



这样就求得式(7.54)是对海森矩阵的近似, 但拟牛顿法的迭代公式(7.45)要求的迭代方向是 $\mathbf{H}_k \nabla f(x^k)$ , 那么应该如何将 $\mathbf{B}_{k+1}$ 转化为 $\mathbf{H}_{k+1}$ 呢?

若 $\mathbf{H}_{k+1} = \mathbf{B}_{k+1}^{-1}$ , 则有:

$$\mathbf{H}_{k+1} = \mathbf{B}_{k+1}^{-1} = \left( \mathbf{B}_k + \frac{\mathbf{y}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} - \frac{\mathbf{B}_k \mathbf{s}_k \mathbf{s}_k^T \mathbf{B}_k}{\mathbf{s}_k^T \mathbf{B}_k \mathbf{s}_k} \right)^{-1}$$

利用Sherman-Morrison公式<sup>[6,7]</sup>, 可以求解出 $\mathbf{B}_{k+1}^{-1}$ 的迭代公式如下:

$$\begin{aligned} \mathbf{B}_{k+1}^{-1} &= \left( \mathbf{I} - \frac{\mathbf{s}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} \right) \mathbf{B}_k^{-1} \left( \mathbf{I} - \frac{\mathbf{y}_k \mathbf{s}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} \right) + \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} \\ \Leftrightarrow \mathbf{H}_{k+1} &= \left( \mathbf{I} - \frac{\mathbf{s}_k \mathbf{y}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} \right) \mathbf{H}_k \left( \mathbf{I} - \frac{\mathbf{y}_k \mathbf{s}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} \right) + \frac{\mathbf{s}_k \mathbf{s}_k^T}{\mathbf{y}_k^T \mathbf{s}_k} \end{aligned} \quad (7.62)$$

#### 7.5.4 L-BFGS算法

BFGS算法最大的问题是存在巨大的内存消耗, 每一步都需要保存 $\mathbf{H}_k$ 。当 $f(x)$ 是 $n$ 元函数时,  $\mathbf{H}_k$ 是一个 $n \times n$ 维的大矩阵; 当 $n = 10$ 万时, 需要使用大约80G的空间存储 $\mathbf{H}_k$ , 由此, 研究人员设计出了BFGS算法的一种改进方案L-BFGS算法, L-BFGS全称为Limited-memory BFGS, 它是对BFGS算法的一种近似, 核心思想是: 算法不再存储完整的矩阵 $\mathbf{H}_k$ , 而是保留向量 $\{\mathbf{s}_k\}$ 、 $\{\mathbf{y}_k\}$ 。并且 $\{\mathbf{s}_k\}$ 和 $\{\mathbf{y}_k\}$ 也不是全部保留, 而是保留最近的 $m$ 个元素值。也就是说, 在第 $k+1$ 次迭代时, L-BFGS算法使用 $(\mathbf{s}_{k-m+1}, \mathbf{s}_{k-m+2}, \dots, \mathbf{s}_k)^T$ 和 $(\mathbf{y}_{k-m+1}, \mathbf{y}_{k-m+2}, \dots, \mathbf{y}_k)^T$ 来模拟牛顿方向 $\mathbf{H}_k \nabla f(x^k)$ 。

首先对式(7.62)进行改写, 设:

$$\rho_k = \frac{1}{\mathbf{y}_k^T \mathbf{s}_k}, \quad \mathbf{V}_k = \mathbf{I} - \rho_k \mathbf{y}_k \mathbf{s}_k^T \quad (7.63)$$

注意, 由于 $\mathbf{y}_k^T \mathbf{s}_k$ 的结果是一个常数值, 故有 $(\rho_k)^T = \rho_k$ 成立, 把式(7.63)代入式(7.62)可得:

$$\mathbf{H}_{k+1} = \mathbf{V}_k^T \mathbf{H}_k \mathbf{V}_k + \rho_k \mathbf{s}_k \mathbf{s}_k^T \quad (7.64)$$

取 $\mathbf{H}_0 = \mathbf{I}$ , 依次代入式(7.64)可得:

$$\mathbf{H}_1 = \mathbf{V}_0^T \mathbf{H}_0 \mathbf{V}_0 + \rho_0 \mathbf{s}_0 \mathbf{s}_0^T$$

$$\mathbf{H}_2 = \mathbf{V}_1^T \mathbf{H}_1 \mathbf{V}_1 + \rho_1 \mathbf{s}_1 \mathbf{s}_1^T$$

$$\begin{aligned}
&= V_1^T (V_0^T H_0 V_0 + \rho_0 s_0 s_0^T) V_1 + \rho_1 s_1 s_1^T \\
&= V_1^T V_0^T H_0 V_0 V_1 + V_1^T \rho_0 s_0 s_0^T V_1 + \rho_1 s_1 s_1^T \\
H_3 &= V_2^T H_2 V_2 + \rho_2 s_2 s_2^T \\
&= V_2^T (V_1^T V_0^T H_0 V_0 V_1 + V_1^T \rho_0 s_0 s_0^T V_1 + \rho_1 s_1 s_1^T) V_2 + \rho_2 s_2 s_2^T \\
&= V_2^T V_1^T V_0^T H_0 V_0 V_1 V_2 + V_2^T V_1^T \rho_0 s_0 s_0^T V_1 V_2 + V_2^T \rho_1 s_1 s_1^T V_2 + \rho_2 s_2 s_2^T
\end{aligned}$$

由此，可以列出 $\mathbf{H}_k$ 的一般形式：

$$\begin{aligned}
\mathbf{H}_k &= (V_{k-1}^T V_{k-2}^T \dots V_0^T) H_0 (V_0 V_1 \dots V_{k-1}) \\
&\quad + (V_{k-1}^T V_{k-2}^T \dots V_1^T) \rho_0 s_0 s_0^T (V_1 V_2 \dots V_{k-1}) \\
&\quad + (V_{k-1}^T V_{k-2}^T \dots V_2^T) \rho_1 s_1 s_1^T (V_2 V_3 \dots V_{k-1}) \\
&\quad + \dots \\
&\quad + (V_{k-1}^T) \rho_{k-2} s_{k-2} s_{k-2}^T (V_{k-1}) + \rho_{k-1} s_{k-1} s_{k-1}^T \quad (7.65)
\end{aligned}$$

由于 $\{\mathbf{s}_k\}$ 和 $\{\mathbf{y}_k\}$ 只保留最近的 $m$ 个元素值，式(7.65)只能计算 $H_1, H_2, \dots, H_m$ 。要计算 $H_{m+1}$ ，需要丢弃 $s_0$ 和 $y_0$ 。同理，要计算 $H_{m+2}$ ，要丢弃 $\{s_0, s_1\}$ 和 $\{y_0, y_1\}$ ，依此类推，当 $k > m$ 时，可以把式(7.65)化简为：

$$\begin{aligned}
\mathbf{H}_k &= (V_{k-1}^T V_{k-2}^T \dots V_{k-m}^T) H_{k-m} (V_{k-m} V_{k-m+1} \dots V_{k-1}) \\
&\quad + (V_{k-1}^T V_{k-2}^T \dots V_{k-m+1}^T) \rho_{k-m} s_{k-m} s_{k-m}^T (V_{k-m+1} V_{k-m+2} \dots V_{k-1}) \\
&\quad + (V_{k-1}^T V_{k-2}^T \dots V_{k-m+2}^T) \rho_{k-m+1} s_{k-m+1} s_{k-m+1}^T (V_{k-m+2} V_{k-m+3} \dots V_{k-1}) \\
&\quad + \dots \\
&\quad + (V_{k-1}^T) \rho_{k-2} s_{k-2} s_{k-2}^T (V_{k-1}) + \rho_{k-1} s_{k-1} s_{k-1}^T \quad (7.66)
\end{aligned}$$

式(7.66)展示了如何通过 $\{\mathbf{s}_k\}$ 和 $\{\mathbf{y}_k\}$ 的最近 $m$ 个元素值，以及 $H_{k-m}$ 来求解 $\mathbf{H}_k$ 的值，但因为仍需要保留 $H_{k-m}$ 的值，总的空间复杂度仍然是 $O(n^2)$ 。事实上，L-BFGS算法的核心原理并不是利用 $\mathbf{H}_k$ 来近似海森逆矩阵 $(\nabla^2 f(\mathbf{x}^k))^{-1}$ ，而是利用 $(s_{k-m}, s_{k-m+1}, \dots, s_{k-1})^T$ 和 $(y_{k-m}, y_{k-m+1}, \dots, y_{k-1})^T$ 来模拟牛顿 $\mathbf{H}_k \nabla f(\mathbf{x}^k)$ ，即：

$$\begin{aligned}
H_k \nabla f(\mathbf{x}^k) &= (V_{k-1}^T V_{k-2}^T \dots V_{k-m}^T) H_{k-m} (V_{k-m} V_{k-m+1} \dots V_{k-1}) \nabla f(\mathbf{x}^k) + \\
&\quad (V_{k-1}^T V_{k-2}^T \dots V_{k-m+1}^T) \rho_{k-m} s_{k-m} s_{k-m}^T (V_{k-m+1} V_{k-m+2} \dots V_{k-1}) \nabla f(\mathbf{x}^k) +
\end{aligned}$$

$$(V_{k-1}^T V_{k-2}^T \dots V_{k-m+2}^T) \rho_{k-m+1} s_{k-m+1} s_{k-m+1}^T (V_{k-m+2} V_{k-m+3} \dots V_{k-1}) \nabla f(x^k) + \dots + (V_{k-1}^T) \rho_{k-2} s_{k-2} s_{k-2}^T (V_{k-1}) \nabla f(x^k) + \rho_{k-1} s_{k-1} s_{k-1}^T \nabla f(x^k) \quad (7.67)$$

首先给出L-BFGS算法在第 $k$ 次迭代时的算法流程如图7.12所示。

算法 7.3 L-BFGS算法流程

---

```

令  $q = \nabla f(x^k) \in \mathbf{R}^n$ 
for  $i = 1, 2, \dots, m$  do
     $a_i = \rho_{k-i} s_{k-i}^T q$ 
     $q = q - a_i y_{k-i}$ 
end
 $r = H_{k-m} q$ 
for  $i = m, m-1, \dots, 1$  do
     $\beta = \rho_{k-i} y_{k-i}^T r$ 
     $r = r + s_{k-i} (a_i - \beta)$ 
end

```

}

前向循环

}

后向循环

---

图7.12 L-BFGS的算法流程

观察前向循环，可知：

$$\begin{aligned}
 q_i &= q_{i-1} - \rho_{k-i} s_{k-i}^T q_{i-1} y_{k-i} = (I - \rho_{k-i} y_{k-i} s_{k-i}^T) q_{i-1} \\
 &= V_{k-i} q_{i-1} = V_{k-i} V_{k-(i-1)} q_{i-2} = \dots = \\
 &= (V_{k-i} V_{k-(i-1)} \dots V_{k-1}) q_0 \\
 &= (V_{k-i} V_{k-(i-1)} \dots V_{k-1}) \nabla f(x^k)
 \end{aligned}$$

因此在前向循环结束的时候，将得到：

$$q_m = (V_{k-m} V_{k-(m-1)} \dots V_{k-1}) \nabla f(x^k) \quad (7.68)$$

观察式(7.68)，我们发现它正是式(7.67)第一项的右半部分。而每一个 $a_i$ 正是式(7.67)中除去第一项外的右半部分。

$$a_i = \rho_{k-i} s_{k-i}^T q_{i-1} = \rho_{k-i} s_{k-i}^T (V_{k-i+1} V_{k-i+2} \dots V_{k-1}) \nabla f(x^k) \quad (7.68)$$

最后，考察后向循环：

$$\begin{aligned}
r_i &= r_{i+1} + s_{k-i}(a_i - \rho_{k-i}y_{k-i}^T r_{i+1}) = (I - \rho_{k-i}s_{k-i}y_{k-i}^T)r_{i+1} + s_{k-i} \times a_i \\
&= V_{k-i}^T r_{i+1} + s_{k-i} \times a_i \quad (7.69)
\end{aligned}$$

先来观察 $r_1$ ：

$$\begin{aligned}
r_1 &= V_{k-1}^T r_2 + s_{k-1} \times a_1 \\
&= V_{k-1}^T r_2 + s_{k-1} \rho_{k-1} s_{k-1}^T \nabla f(x^k) \\
&= V_{k-1}^T V_{k-2}^T r_3 + V_{k-1}^T s_{k-2} a_2 + s_{k-1} \rho_{k-1} s_{k-1}^T \nabla f(x^k) \\
&= V_{k-1}^T V_{k-2}^T r_3 + V_{k-1}^T s_{k-2} \rho_{k-2} s_{k-2}^T V_{k-1}^T \nabla f(x^k) + s_{k-1} \rho_{k-1} s_{k-1}^T \nabla f(x^k) \\
&= \dots \\
&= (V_{k-1}^T V_{k-2}^T \dots V_{k-m}^T) r_{m+1} + (V_{k-1}^T V_{k-2}^T \dots V_{k-m+1}^T) s_{k-m} a_m + \dots \\
&\quad + V_{k-1}^T s_{k-2} \rho_{k-2} s_{k-2}^T V_{k-1}^T \nabla f(x^k) + s_{k-1} \rho_{k-1} s_{k-1}^T \nabla f(x^k) \\
&= (V_{k-1}^T V_{k-2}^T \dots V_{k-m}^T) H_{k-m} (V_{k-m} V_{k-(m-1)} \dots V_{k-1}) \nabla f(x^k) \\
&\quad + (V_{k-1}^T V_{k-2}^T \dots V_{k-m+1}^T) s_{k-m} \rho_{k-m} s_{k-m}^T (V_{k-m+1} V_{k-(m-1)} \dots V_{k-1}) \nabla f(x^k) + \dots \\
&\quad + V_{k-1}^T s_{k-2} \rho_{k-2} s_{k-2}^T V_{k-1}^T \nabla f(x^k) + s_{k-1} \rho_{k-1} s_{k-1}^T \nabla f(x^k) \quad (7.70)
\end{aligned}$$

$r_1$ 的值与式(7.67)完全一致，因此，当后向循环完成后，得到的 $r_1$ 就是当前第 $k$ 次迭代的牛顿方向 $H_k \nabla f(x^k)$ 。

## 7.6 约束最优化条件

本节探讨带约束条件的最优化问题，形式化地说，把下面的最优化问题称为带约束条件的最优化问题。

$$\begin{aligned}
&\min f(x), x \in \mathbf{R}^n \\
&s.t. \quad h_j(x) = 0, j = 1, 2, \dots, m \\
&\quad \quad g_i(x) \geq 0, i = 1, 2, \dots, n \quad (7.71)
\end{aligned}$$

其中，由 $h_j(x) = 0$ 和 $g_i(x) \geq 0$ 所包含的区域，称为可行域。一般情况下，求解约束优化问题要比无约束问题困难很多，因为对于无约束优化，考虑的重点是寻找下降的方向 $d$ ，对于带约束条件的最优化问题，必须保证在方向 $d^*$ 有可行点，也就是在方向 $d^*$ 上至少有一小段区域能被包含在可行域内，因此，求解带约束条件的最优化问题

本质上是要寻找下降的可行方向 $d^*$ 。

### 1. 等式约束的最优化必要条件

首先来考察等式约束的最优化条件，即：

$$\begin{aligned} \min f(x) \quad & x \in \mathbf{R}^n \\ \text{s.t. } h_j(x) = 0 \quad & j = 1, 2, \dots, m \end{aligned} \quad (7.72)$$

其中， $f(x)$ 与 $h_j(x)$ 均为可微函数，而约束条件 $h_j(x) = 0$ 则相当于把可行域限于曲面，记为曲面 $\Omega_j$ 。设式(7.72)的最优解为 $x^* = (x_1^*, x_2^*, \dots, x_n^*)^T$ ，过点 $x^*$ 在曲面 $\Omega_j$ 上作一条曲线 $l$ ，曲线方程记为：

$$x_1 = x_1(t)$$

$$x_2 = x_2(t)$$

$$\dots \dots$$

$$x_n = x_n(t)$$

由于曲线位于曲面 $\Omega_j$ 上，故必有 $h_j(x_1(t), x_3(t), \dots, x_n(t)) = 0$ 成立。设 $x^* = (x_1(t^*), x_2(t^*), \dots, x_n(t^*))^T$ ，由于 $x^*$ 是 $f(x)$ 的极小值点，因此 $t^*$ 也是一元函数 $f(t)$ 的极小值点，由极值的必要条件可得：

$$\begin{aligned} \frac{df(t^*)}{dt} &= 0 \\ \Leftrightarrow \frac{df(x^*)}{dx_1} x_1'(t^*) + \frac{df(x^*)}{dx_2} x_2'(t^*) + \dots + \frac{df(x^*)}{dx_n} x_n'(t^*) &= 0 \end{aligned} \quad (7.73)$$

设向量 $\nabla f(x^*) = \left( \frac{df(x^*)}{dx_1}, \frac{df(x^*)}{dx_2}, \dots, \frac{df(x^*)}{dx_n} \right)^T$ ，曲线 $l$ 在点 $x^*$ 处的切线方向为 $(x_1'(t^*), x_2'(t^*), \dots, x_n'(t^*))^T$ ，由式(7.73)以及曲线 $l$ 的任意性可知， $\nabla f(x^*)$ 必垂直于 $\Omega_j$ 在点 $x^*$ 的切平面，而 $\nabla h_j(x^*)$ 是曲面 $\Omega_j$ 在点 $x^*$ 的法向量，故向量 $\nabla f(x^*)$ 与向量 $\nabla h_j(x^*)$ 共线，即必存在实数 $\mu^*$ ，使得：

$$\nabla f(x^*) = \mu^* \times \nabla h_j(x^*) \quad (7.74)$$

成立。依此类推，对于式(7.72)的所有 $m$ 个等式约束，有：

$$\nabla f(x^*) = \sum_{j=1}^m \mu_j^* \times \nabla h_j(x^*) \quad (7.75)$$

为了方便理解式(7.75)，引入函数：

$$L(x, \mu_1, \mu_2, \dots, \mu_m) = f(x) - \sum_{j=1}^m \mu_j \times h_j(x) \quad (7.76)$$

式(7.76)被称为**拉格朗日函数**， $\mu = (\mu_1, \mu_2, \dots, \mu_m)^T$ 称为拉格朗日乘子，由拉格朗日函数，可以把式(7.75)等价表示为： $\nabla_x L(x, \mu) = 0$ 。由此，得到下面关于**等式约束最优化存在的必要条件**。

定理：对于式(7.72)的等式约束最优化问题，若 $f(x)$ 、 $h_j(x)$  ( $j = 1, 2, \dots, m$ )均一阶可微， $\nabla h_1(x^*)$ 、 $\nabla h_2(x^*)$ 、 $\dots$ 、 $\nabla h_m(x^*)$ 线性无关，则 $x^*$ 是 $f(x)$ 的最小值点的必要条件是存在 $m$ 个拉格朗日乘子 $\mu^* = (\mu_1^*, \mu_2^*, \dots, \mu_m^*)^T$ 使得：

$$\nabla_x L(x, \mu^*) = \nabla f(x^*) - \sum_{j=1}^m \mu_j^* \times \nabla h_j(x^*) = 0 \quad (7.77)$$

## 2. 不等式约束的最优化必要条件

进一步来考察既有不等式约束，也有等式约束的最优化问题，即：

$$\min f(x) \quad x \in \mathbf{R}^n \quad (7.78)$$

$$\text{s.t. } g_i(x) \geq 0 \quad i = 1, 2, \dots, n \quad (7.79)$$

$$h_j(x) = 0 \quad j = 1, 2, \dots, m \quad (7.80)$$

首先构造函数：

$$L(x, \alpha, \beta) = f(x) - \sum_{i=1}^n \alpha_i \times g_i(x) - \sum_{j=1}^m \beta_j \times h_j(x) \quad (7.81)$$

与式(7.76)相对应，式(7.81)被称为广义的拉格朗日函数，其中 $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n)^T$ 和 $\beta = (\beta_1, \beta_2, \dots, \beta_m)^T$ 称为拉格朗日乘子。现在考察：

$$\theta(x) = \max_{\alpha, \beta} L(x, \alpha, \beta) \quad (7.82)$$

观察式(7.82)，考察 $\theta(x)$ ：

对于任意的 $i = 1, 2, \dots, n$ ，当 $g_i(x) < 0$ ，这时若 $\alpha_i$ 取无穷大的正数，必有 $\theta(x) = \infty$ ；

同理, 对于任意的  $j = 1, 2, \dots, m$ , 当  $h_j(x) \neq 0$ , 必有  $\theta(x) = \infty$ 。

$$\theta(x) = \begin{cases} f(x) & \text{当满足约束式(7.79)和式(7.80)} \\ \infty & \text{当不满足约束条件式(7.79)和式(7.80)} \end{cases} \quad (7.83)$$

因此  $\min f(x) = \min \theta(x) = \min(\max_x L(x, \alpha, \beta))$  成立, 当  $x^*$  为  $f(x)$  的最优值点时,  $x^*$  同样是  $\max_x L(x, \alpha, \beta)$  的最优值点, 故必有:

$$\nabla_x L(x^*, \alpha^*, \beta^*) = \nabla f(x^*) - \sum_{i=1}^n \alpha_i \times \nabla g_i(x^*) - \sum_{j=1}^m \beta_j \times \nabla h_j(x^*) = 0$$

成立。

此外, 对于所有的非线性约束  $g_i(x) \geq 0 (i = 1, 2, \dots, n)$ , 若  $g_i(x) = 0$ , 那么对于  $x^*$  而言,  $g_k(x) > 0 (k = 1, 2, \dots, i-1, i+1, \dots, n)$  并没有起到实质作用, 故式(7.78)等价于:

$$\min f(x) \quad x \in \mathbf{R}^n \quad (7.84)$$

$$s. t. \quad g_i(x) = 0 \quad (7.85)$$

$$h_j(x) = 0 \quad j = 1, 2, \dots, m \quad (7.86)$$

为了统一起见, 把  $g_i(x) \geq 0 (i = 1, 2, \dots, n)$  用  $\alpha_i \times g_i(x) = 0 (\alpha_i \geq 0)$  来等价表示。

综上所述, 得到如下判断非线性约束存在最优解的必要条件定理。

定理: 对于式(7.78)的不等式约束最优化问题, 若  $f(x)$ 、 $g_i(x) (i = 1, 2, \dots, n)$ 、 $h_j(x) (j = 1, 2, \dots, m)$  均为一阶可微,  $\nabla h_j(x^*)$ ,  $\nabla g_i(x^*)$  线性无关, 则  $x^*$  是  $f(x)$  的最小值点的必要条件是存在拉格朗日乘子  $\alpha^* = (\alpha_1^*, \alpha_2^*, \dots, \alpha_m^*)^T$ , 以及  $\beta^* = (\beta_1^*, \beta_2^*, \dots, \beta_m^*)^T$ , 使得:

$$\nabla_x L(x^*, \alpha^*, \beta^*) = \nabla f(x^*) - \sum_{i=1}^n \alpha_i^* \times g_i(x^*) - \sum_{j=1}^m \beta_j^* \times h_j(x^*) = 0 \quad (7.87)$$

$$\alpha_i^* \times g_i(x^*) = 0 \quad (7.88)$$

$$\alpha_i^* \geq 0 \quad (7.89)$$

我们把式(7.87)、式(7.88)和式(7.89)称为KKT条件, 把满足KKT条件的点称为KKT点, 上面的定理表明, 若式(7.78)的不等式约束最优化存在最优解, 那么最优点一定是KKT点。

## 参考文献：

---

- [1] Richard Courant, Fritz John. Introduction to Calculus and Analysis.
- [2] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks: The Official Journal of the International Neural Network Society*, 12(1), 145–151.
- [3] Nesterov, Y. (1983). A method for unconstrained convex minimization problem with the rate of convergence  $o(1/k^2)$ . *Doklady ANSSSR*, vol. 269, pp. 543–547.
- [4] John Duchi, Elad Hazan, Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*. 2011.
- [5] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method, arXiv:1212. 5701.
- [6] Sherman, Jack; Morrison, Winifred J. (1949). Adjustment of an Inverse Matrix Corresponding to Changes in the Elements of a Given Column or a Given Row of the Original Matrix (abstract). *Annals of Mathematical Statistics*. 20: 621.
- [7] Sherman, Jack; Morrison, Winifred J. (1950). Adjustment of an Inverse Matrix Corresponding to a Change in One Element of a Given Matrix. *Annals of Mathematical Statistics*. 21 (1): 124–127.
- [8] Sebastian Ruder. An overview of gradient descent optimization algorithms. 2016.



---

# 第 3 部分

## 理论与应用篇

---

# 8

## 前馈神经网络

在绪论中，我们回顾了深度学习的发展历史，深度学习并不是一门新兴的学科，过去它更多的是以神经网络的形式而存在，并作为机器学习的一个算法分支而发展，因此，当深度学习指代的是深度神经网络（在第11章将看到深度学习的另一种形式，即深度图模型）时，深度学习中的“深”就是指神经网络的层次深度。而前馈神经网络（Feedforward Neural Network）是最早被提出的最简单的神经网络模型，这一章先详细讲解前馈神经网络的相关知识，它们是后面各章的理论基础。

本章包括5部分内容。

**8.1节：生物神经元结构。**神经网络是与大脑研究密切相关的一门学科，其大部分理论知识都来源于对生物大脑的研究，这其中神经元又是神经网络最基本的组成单元。因此，本节将了解生物神经元的相关知识，包括生物神经元的历史、结构及其工作原理。

**8.2节：人工神经元结构。**受生物神经元的启发，我们构造出人工神经元用以模拟生物神经元的工作方式，并将深入分析人工神经元对数据信息的处理流程。

**8.3节：单层感知机（Perceptron）。**单层感知机是最早的神经网络模型，由输入层和输出层构成。

**8.4节：多层感知机（Multilayer Perceptron，简称MLP）。**在单层感知机的基础上，我们将构建更加复杂的带隐藏层的感知机模型。相对于单层感知机，MLP具备了

解决非线性分类问题的能力。

8.5节：激活函数（Activation function）。本节将探讨神经网络的一个很重要组成部分：激活函数，可以说激活函数是神经网络能够解决复杂问题的核心，我们会讲解当前常用的激活函数模型，并考察不同的激活函数各自具有的特点。

## 8.1 生物神经元结构

神经网络起源于对生物神经元（Biological Neurons，简称BN）的研究，生物学家对BN的研究由来已久，早在1904年，生物学家就已经知晓神经元的组成结构，人的大脑中大约有100亿到1000亿个神经元，每个神经元大约会与其他1万个神经元相连，如下图8.1所示。

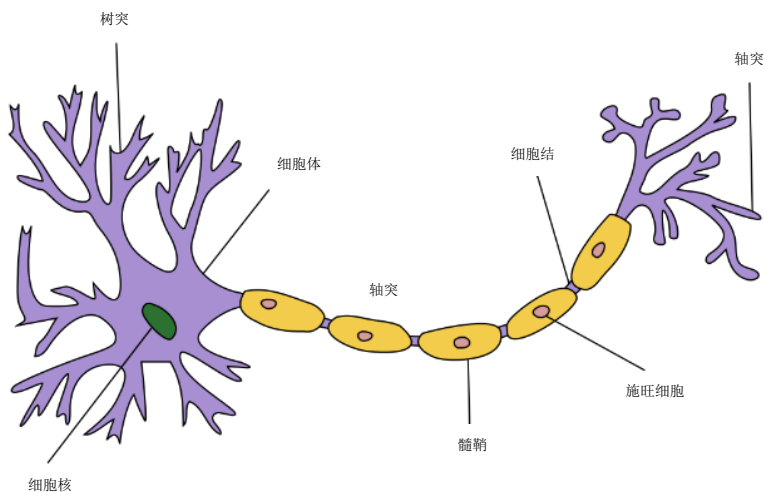


图8.1 生物神经元结构图（图片来源于网络）

从图8.1中可以看出每一个生物神经元主要由细胞体、树突、轴突和突触4个部分构成。

### 1. 细胞体

细胞体是神经元的主体，由细胞核、细胞质和细胞膜3部分组成。细胞体的外部是细胞膜，将膜内外细胞液分开。由于细胞膜对细胞液中的不同离子具有不同的通透性，这使得膜内外存在着离子浓度差，从而出现内负外正的静息电位，这种电位差称为膜电位。

## 2. 树突

从细胞体向外延伸出许多突起的神经纤维，负责接收来自其他神经元的输入信号，相当于细胞体的输入端。

## 3. 轴突

由细胞体伸出的最长的一条突起称为轴突。轴突比树突长而细。轴突也叫神经纤维，末端处有很多细的分支称为神经末梢，每一条神经末梢可以向四面八方传出信号，相当于细胞体的输出端。

## 4. 突触

突触是一个神经元通过其轴突的神经末梢和另一个神经元的细胞体或树突进行通信连接。突触使神经细胞的膜电位发生变化，且电位的变化是可以累加的。

综上所述，一个神经元的输入端有多个树突，主要是用来接受输入信息。输入信息经过突触处理，将输入信息累加，当处理后的输入信息大于某一个特定的阈值，就会把信息通过轴突传播出去，这时称神经元被激活，相反，当处理后的输入信息小于阈值时，神经元就处于抑制状态，它不会向其他神经元传递数据或者传递很少的信息数据。

## 8.2 人工神经元结构

受生物神经元的启发，1943年心理学家McCulloch与数学家Pitts<sup>[1]</sup>提出了人工神经元模型（Artificial Neuron，简称AN），如图8.2所示，人们也常用他们两个人的名字的首字母来命名这个人工神经元模型，称之为M-P模型，这种人工神经元模型也一直沿用至今。

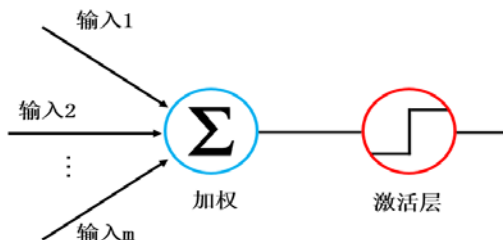


图8.2 人工神经元结构图

AN是神经网络中的最小信息处理单元，与生物神经元类似，AN对数据的处理分为两个阶段：第一阶段是接收来自其他 $m$ 个神经元传递过来的输入信号，这些输入信号通过与相应权重进行加权求和传递给下一阶段，这一阶段被称为预激活阶段（pre-activation），即满足：

$$h = b + \left( \sum_{i=1}^m w_i \times x_i \right) \tag{8.1}$$

第二阶段是把预激活的加权结果传递给激活函数，一般来说，经过激活函数的处理后，预激活的数值将被压缩到一个范围区间内，数值的大小将决定神经元到底是处于活跃状态还是抑制状态，最后将输出结果传递给下一层的神经元：

$$a = f(h) \tag{8.2}$$

其中， $f$ 是激活函数，有关激活函数的相关知识将在稍后章节来探讨。下面通过图8.3来简单概括人工神经元与生物神经元的对应关系。

生物神经元	人工神经元
细胞核	神经元
树突	输入
轴突	输出
突触	权重

图8.3 人工神经元与生物神经元的对比

### 8.3 单层感知机

单层感知机是最早被提出的最简单的神经网络模型，它仅由输入层和输出层构成，在1957年由计算机科学家Rosenblatt提出<sup>[7]</sup>，网络结构如图 8.4所示。

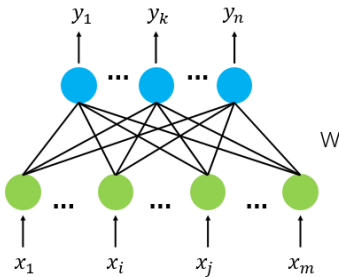


图8.4 单层感知机模型

Perceptron由 $m$ 个输入神经元和  $n$  个输出神经元构成，是一个全连接的二分图结构，单层感知机首先在输入层接收输入数据，把输入数据与相应权重参数累加。

$$h_k = b_k + \left(\sum_{i=1}^m w_{ik} \times x_i\right), \quad k = 1, 2, \dots, n \tag{8.3}$$

然后把累加结果输入到激活函数中，单层感知机的输出层可以采用Logistic或者softmax函数作为最后的结果输出：

$$y_k = \text{softmax}(h_k), \quad k = 1, 2, \dots, n \tag{8.4}$$

下面分析感知机的工作原理。单层感知机本质上是在高维空间中，构造出合理的边界超平面( $b + \sum w_i x_i$ )，把不同类别的数据集分离，因此，对于线性可分，或者近似线性可分的数据集有很好的效果，如图8.6所示。

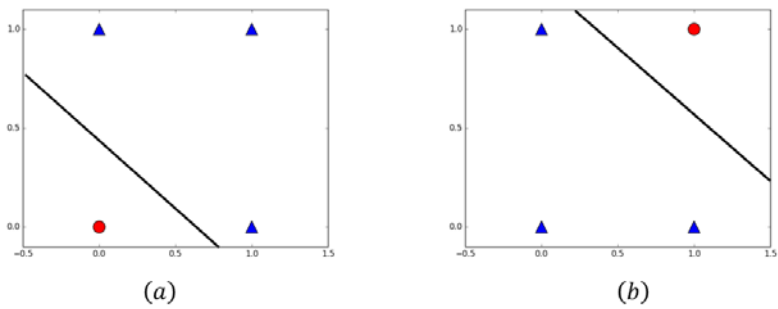


图8.5 利用单层感知机对“与”和“或”二值问题进行线性划分

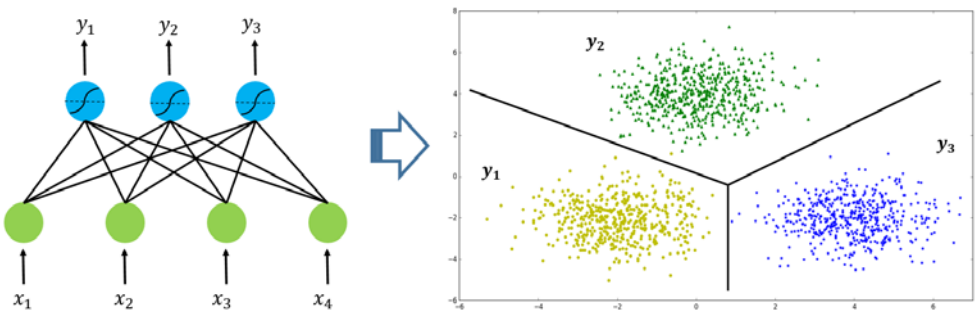


图8.6 利用单层感知机划分线性可分数据集

但对于线性不可分数据集，由于我们没有办法仅通过一个超平面来分离数据，因此Perceptron的效果不甚理想，也导致单层感知机的学习能力非常有限，如图8.7所示，它甚至不能解决简单的“异或”问题。

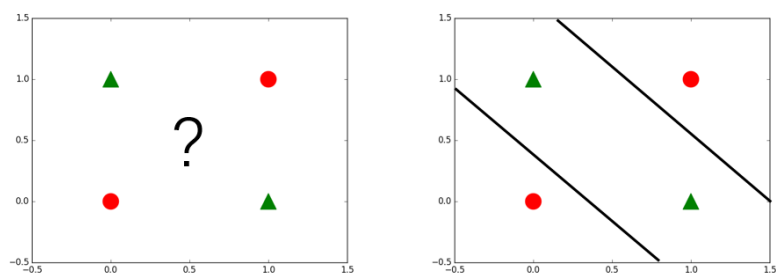


图8.7 “异或”问题是线性不可分的，利用单层感知机无法求解

下面来看一个实际的例子，Minist数据集是一个手写图片识别的数据库，是机器学习领域中的一个经典问题。它的任务是把 $28 \times 28$ 像素的灰度手写数字图片识别为相应的数字，其中数字的范围从0到9。原始的Minist数据集中，每一个训练数据都是大小为 $28 \times 28$ 的灰度图片，也就是输入数据是784维的向量空间，为了直观地观察数据的可视化效果，首先利用t-sne算法把高维空间数据投影到二维空间中，如图8.8所示。

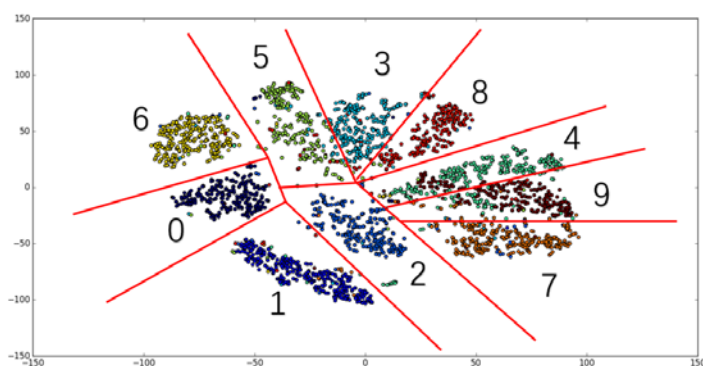


图8.8 利用t-sne算法对Minist测试数据集进行降维的效果图

从图8.8中可以看出，像0、1、6这几个数字，它们的数据与其他数据之间相对独立，因此与其他数字是线性可分的，而3、5、8之间和4、9之间，有些测试数据即使通过肉眼也很难分辨出来，数据之间也呈现出非线性关系。

算法迭代1 000次后，分别计算每一个Label的精确率和召回率，结果如图8.9所示。

Label	准确率	召回率
0	0.956132	0.978571
1	0.964410	0.978855
2	0.931727	0.899225
3	0.907389	0.911881
4	0.929878	0.931772
5	0.909412	0.866592
6	0.937436	0.954071
7	0.932419	0.926070
8	0.872383	0.898357
9	0.914086	0.906838

图8.9 单层感知机对Minist数据集的分类效果

通过图8.10更形象地观察感知机的分类效果，从图中可以看到，对于线性可分的数据集，如数字0、1、6，它们的分类效果明显要高于其他数字，但对于非线性数据的分类效果则不太理想。

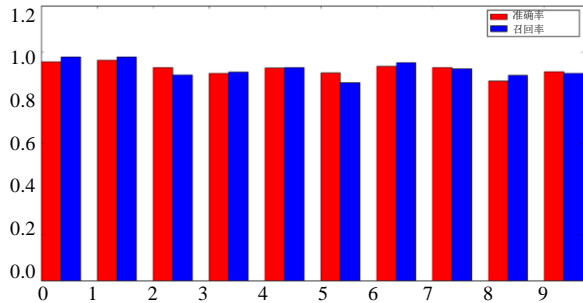


图8.10 对于0、1这几个数字，它们的准确率和召回率都很高；但对于5和8，效果则不理想，主要是由于这些数字之间，在某些样本数据上的相似度极高，而单层感知机无法学习到更深层次的关系

### 8.4 多层感知机

单层感知机因为只在输出层存在激活函数，学习能力非常有限，从8.3节的例子中我们可以看到，单层感知机对于非线性分类数据的效果表现不太理想，要解决这类非线性分类问题，需要在输入层和输出层之间引入隐藏层，让网路有更强大的学习能力，如8.11图所示。



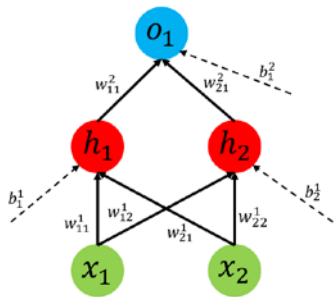


图8.11 包含一个隐藏层，隐藏层中含有两个神经元的多层感知机模型

读者可能会有这样的疑惑，为什么添加隐藏层的神经网络模型能够解决非线性问题呢？或者说深层比浅层网络有哪些方面的改进呢？从理论的角度来说，Kurt Hornik在1991年提出的全局逼近定理（Universal approximation theorem）为我们提供了理论的依据<sup>[3]</sup>：对于含有一个隐藏层的前馈神经网络，如果隐藏层可以由任意多的神经元构成，那么神经网络能够逼近实数范围内的任意连续函数。

从具体的应用角度来看，对于分类问题，隐藏层的作用就是把线性不可分的数据，通过线性变换（预激活阶段）和非线性的激活（激活阶段）的操作，使得数据在输出层变成线性可分。我们来看前面提到的Perceptron无法解决的“异或”问题，MLP的处理过程就是首先把输入数据 $(x_1, x_2)$ 变换为 $(\text{AND}(\sim x_1, x_2), \text{AND}(x_1, \sim x_2))$ ，并以此作为新的输入，把数据变为了线性可分问题，如图8.12所示。

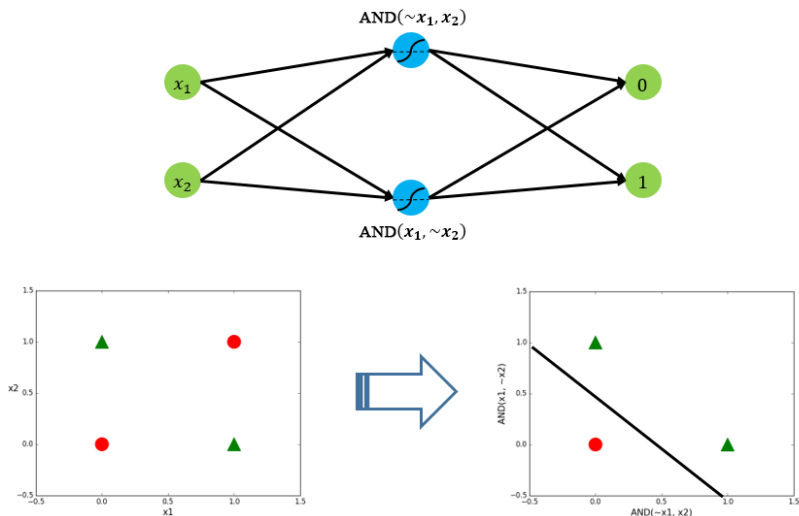


图8.12 利用隐藏层将“异或”问题变换为线性可分问题

通过可视化的具体例子来形象分析MLP是如何构建非线性边界平面的，图8.13是带有一个隐藏层的网络，隐藏层含有两个神经元和网络的权重参数。

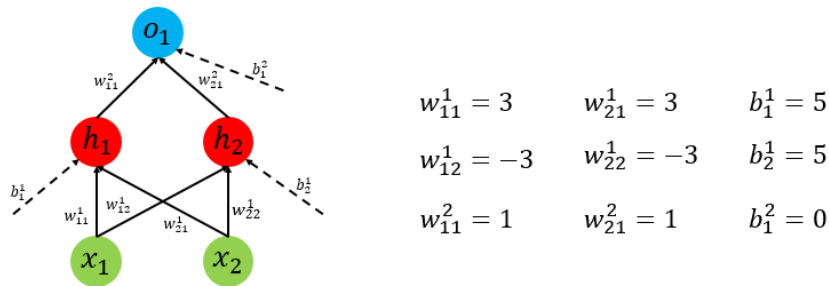


图8.13 含有一个隐藏层、两个隐藏层单元的网络模型，模型的权重参数已知

隐藏层结点的激活函数采用sigmoid函数，图8.14可视化展示了隐藏层函数和输出层函数的效果图，可以看到，该网络经过隐藏层的处理后，在输出层的边界平面实际上由两个超平面共同组成（参见图8.7），具备了初级的非线性学习能力。

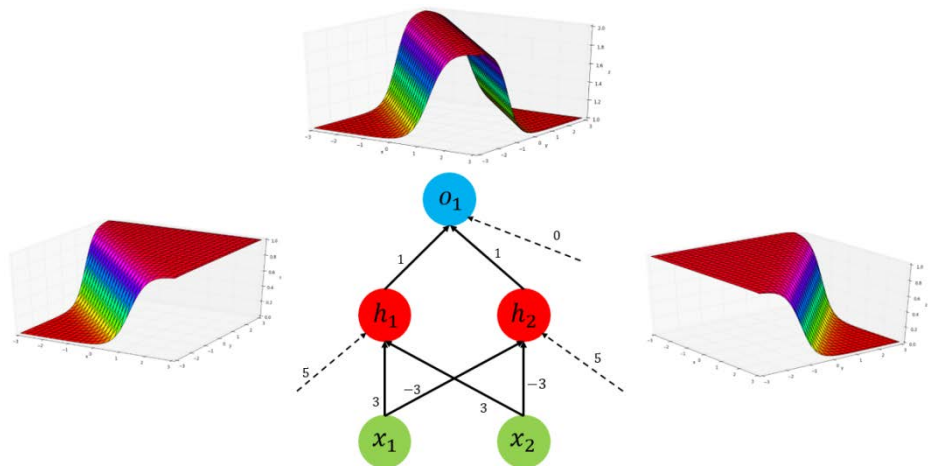


图8.14 当隐藏层含有两个神经元时，每个神经元对应于不同的Logistic输出，经过线性组合后，到达输出层，产生非线性的分界平面

再来看一个更复杂的例子，当隐藏层中含有4个神经元时，与上述例子类似，假设已经给定了网络的权重参数，如图8.15所示。

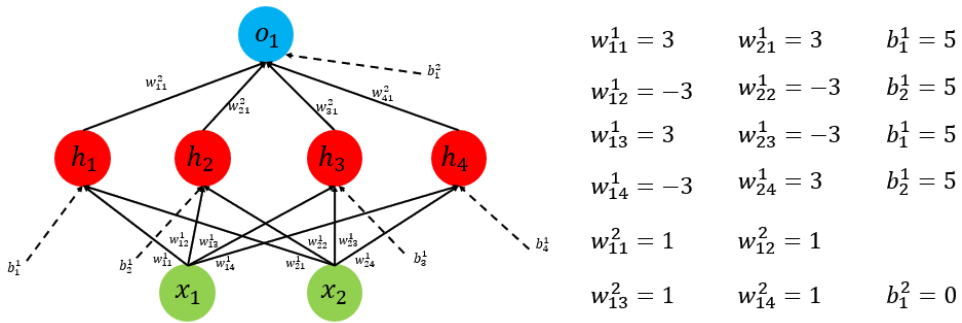


图8.15 含有一个隐藏层，4个隐藏层神经元的多层感知机

图8.16是隐藏层函数和输出层函数对应的可视化效果图，不难发现，和前面只有两个隐藏层神经元的模型相比，当前的网络模型处理能力更加强大，在输出端，网络能够学习到更为复杂的非线性边界平面。

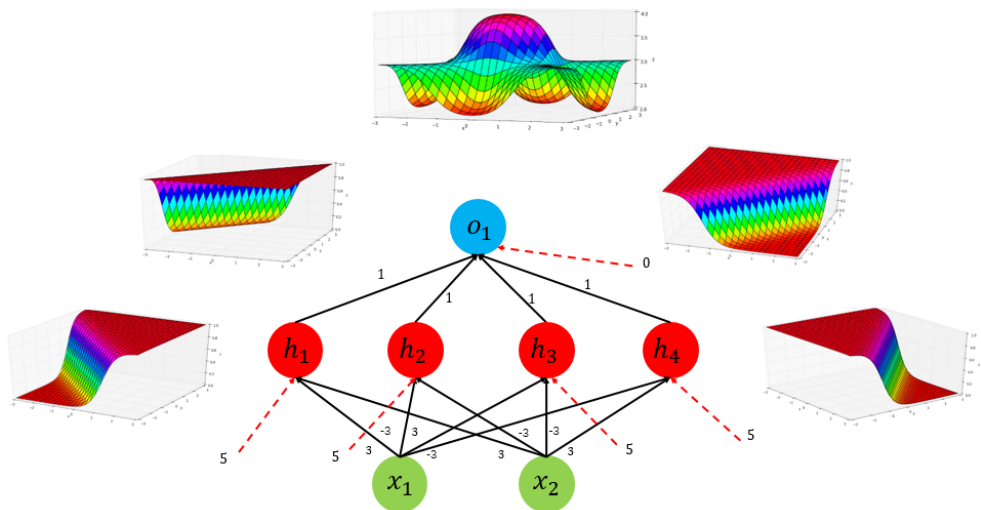


图8.16 当隐藏层含有四个神经元时，每个神经元对应于不同的Logistic输出，经过线性组合后，到达输出层，产生非线性的分界平面，该边界实际上是一个圆

通过上面的分析，我们可以看到，随着隐藏层神经元越来越多，网络具有的学习能力就越强大。来回顾上一节提到的Minist数据集分类问题，现在构建MLP来重新解决这个问题，创建了含有一个隐藏层的MLP模型，隐藏层含有500个神经元，算法经过1 000次迭代训练后，得到的结果如图8.17所示。

Label	准确率	召回率
0	0.983806	0.991837
1	0.989446	0.991189
2	0.982541	0.981589
3	0.980296	0.985149
4	0.986694	0.981670
5	0.987500	0.974215
6	0.980249	0.984342
7	0.980601	0.983463
8	0.983556	0.982546
9	0.983101	0.980178

图8.17 带一个隐藏层，隐藏层含有500个神经元的分类效果

比较图8.17与图8.9的结果数据，我们发现两个有趣的结果。

- 利用MLP分类的结果，无论是准确率还是召回率等指标都有了大幅度的提升，进一步印证了MLP比单层感知机具有更强的学习能力，如图8.18所示。
- 另一方面，各个输出类型的结果没有明显的区别，不管是线性可分还是不可分的类型，它们的结果都非常接近，这说明MLP有效解决了非线性边界问题，是否线性可分已经不是制约性能进一步提升的关键因素。

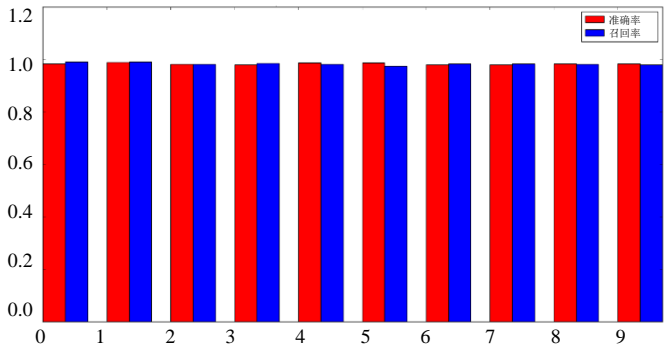


图8.18 MLP的分类效果

前面我们深入分析了多层感知机的原理，以及其强大的学习能力，那么一个想法就是，是否神经网络模型的隐藏层越深，隐藏层的神经元越多，学习能力就越强呢？从理论上来说，神经网络中包含的隐藏层越深，隐藏层包含的神经元越多，它能提取的特征也就越丰富，也就具备更强大的学习能力，但实际的效果却不是这样，随着网络变得越来越深，神经网络的性能在到达一个极值后（一般两到三层），很快就停止

提升,甚至出现下降的情况,我们将在下一章深入分析这个问题。

## 8.5 激活函数

激活函数是神经网络设计的一个核心单元。生物学的理论认为,大脑中每一个神经元具有不同的处理功能,当处理某一个任务时,某些神经元会异常活跃,相反,某些神经元则几乎不参与工作。与之相对应,在神经网络中,把处在活跃状态的神经元称为激活态,处在非活跃状态的神经元称为抑制态,激活函数赋予了神经元自我学习和适应的能力。

### 8.5.1 激活函数的作用

激活函数的作用是为了在神经网络中引入非线性的学习和处理能力,正如8.3节所介绍的,单层感知机之所以学习能力非常有限,是因为模型只能解决线性可分问题。

而MLP通过引入隐藏层,使得模型能够处理非线性数据,从而有效提高模型的学习能力,但应该注意的是,模型能力的提升并不是因为添加了隐藏层,而是在于隐藏层神经元中引入了激活函数。事实上,对于多层神经网络,如果只添加隐藏层,但没有为隐藏层的神经元添加激活函数,或者激活函数是线性函数,那么MLP是等价于单层感知机的,也就是说,不管叠加了多少的隐藏层,其效果与单层感知机没有区别。

为了验证上面的结论,我们来看下面带有一个隐藏层,隐藏层含有3个神经元的网络模型,为了不失一般性,采用恒等函数 $f(x) = x$ 作为激活函数(事实上,采用任意的线性函数 $f(x) = a \times x + b$ 得到的结果也是一样的)如图8.19所示。

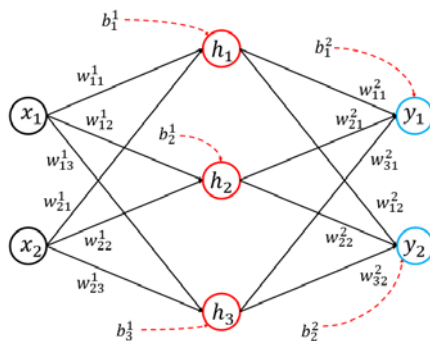


图8.19 带有一个隐藏层的MLP模型,隐藏层神经元的激活函数为恒等函数

其中，隐藏层的输入为：

$$h_1 = w_{11}^1 \times x_1 + w_{21}^1 \times x_2 + b_1^1 \quad (8.5)$$

$$h_2 = w_{12}^1 \times x_1 + w_{22}^1 \times x_2 + b_2^1 \quad (8.6)$$

$$h_3 = w_{13}^1 \times x_1 + w_{23}^1 \times x_2 + b_3^1 \quad (8.7)$$

由于激活函数是恒等函数，因此，经过激活函数的处理后，输入与输出是一致的，即有：

$$a_1 = h_1, a_2 = h_2, a_3 = h_3$$

把式(8.5)、式(8.6)和式(8.7)相结合，得到输出层神经元 $y_1$ 的输入结果为：

$$\begin{aligned} y_1 &= w_{11}^2 \times a_1 + w_{21}^2 \times a_2 + w_{31}^2 \times a_3 + b_1^2 \\ &= w_{11}^2 \times (w_{11}^1 \times x_1 + w_{21}^1 \times x_2 + b_1^1) + w_{21}^2 \times (w_{12}^1 \times x_1 + w_{22}^1 \times x_2 + b_2^1) + \\ &\quad w_{31}^2 \times (w_{13}^1 \times x_1 + w_{23}^1 \times x_2 + b_3^1) + b_1^2 \\ &= (w_{11}^2 \times w_{11}^1 + w_{21}^2 \times w_{12}^1 + w_{31}^2 \times w_{13}^1) \times x_1 + \\ &\quad (w_{11}^2 \times w_{21}^1 + w_{21}^2 \times w_{22}^1 + w_{31}^2 \times w_{23}^1) \times x_2 + \\ &\quad (w_{11}^2 \times b_1^1 + w_{21}^2 \times b_2^1 + w_{31}^2 \times b_3^1 + b_1^2) \\ &= W_{11} \times x_1 + W_{12} \times x_2 + B_1 \end{aligned} \quad (8.8)$$

其中：

$$W_{11} = (w_{11}^2 \times w_{11}^1 + w_{21}^2 \times w_{12}^1 + w_{31}^2 \times w_{13}^1) \quad (8.9)$$

$$W_{12} = (w_{11}^2 \times w_{21}^1 + w_{21}^2 \times w_{22}^1 + w_{31}^2 \times w_{23}^1) \quad (8.10)$$

$$B_1 = (w_{11}^2 \times b_1^1 + w_{21}^2 \times b_2^1 + w_{31}^2 \times b_3^1 + b_1^2) \quad (8.11)$$

同理，可以得到输出层神经元 $y_2$ 的输入结果为：

$$y_2 = (W_{21} \times x_1 + W_{22} \times x_2 + B_2) \quad (8.12)$$

其中：

$$W_{21} = (w_{12}^2 \times w_{11}^1 + w_{22}^2 \times w_{12}^1 + w_{32}^2 \times w_{13}^1) \quad (8.13)$$

$$W_{12} = (w_{12}^2 \times w_{21}^1 + w_{22}^2 \times w_{22}^1 + w_{32}^2 \times w_{23}^1) \quad (8.14)$$

$$B_1 = (w_{12}^2 \times b_1^1 + w_{22}^2 \times b_2^1 + w_{32}^2 \times b_3^1 + b_2^2) \quad (8.15)$$

上面的结论表明，当激活函数为线性函数时，输出层的结果完全由输入层数据决定，隐藏层并没有起到任何作用，所以无论添加多少层，其效果都相当于没有隐藏层的单层感知机模型，如图8.20所示。

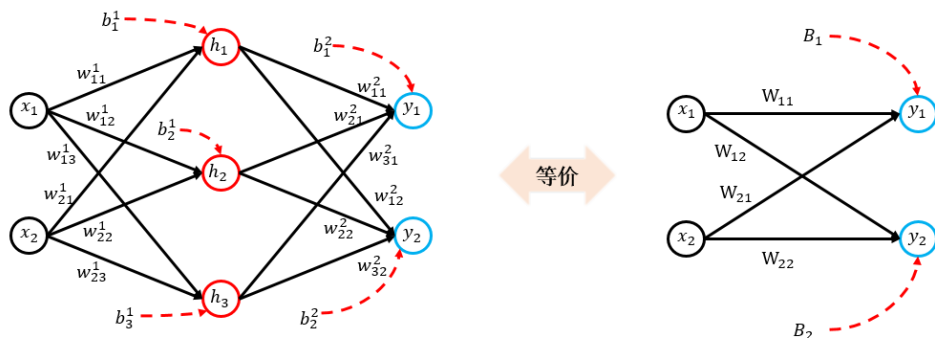


图8.20 当激活函数为线性函数时，MLP模型等价于单层感知机模型

最后，我们来归纳一下激活函数在神经网络中主要起到的作用。

- 非线性的激活函数使得模型能够处理更加复杂的非线性数据集问题，提高了模型的学习能力，这也是引入激活函数最主要的作用。
- 仿照生物神经元的思想，通过激活函数的处理后，神经元被划分为激活态和抑制态，因而，在训练的过程中，能够把起重点作用的神经元置为激活态，而把相对无关的神经元置为抑制态，起到自动特征提取的作用。

### 8.5.2 常用的激活函数

要设计一个合理的激活函数，首先需要满足激活函数的一些性质。

- 非线性：8.5.1节已经分析了线性激活函数的缺点，因此，在设计激活函数时，首先要保证激活函数是非线性的。
- 可微性：在训练网络模型时，基于梯度的模型最优化方法要求激活函数必须是可导的。
- 单调性：单调函数能够保证模型简单。

1. 阶跃函数

阶跃函数是最理想的激活函数，它直接将输入数据映射为0或1，1表示当前神经元处于激活状态，相反，当值为 0，表示当前神经元处于抑制状态，如图8.21所示。

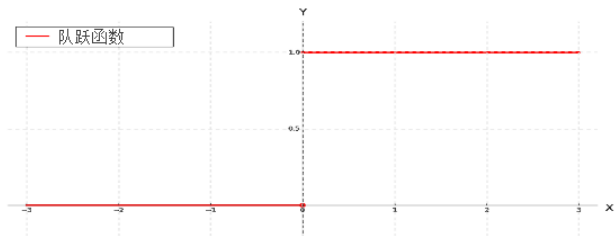


图8.21 阶跃函数

但阶跃函数具有不连续、不光滑、不可导的特点，因此在实际应用中，通常不会采用阶跃函数来作为激活函数，但它为寻找可用的激活函数提供了参考的依据。

2. sigmoid函数和 tanh 函数

sigmoid函数系激活函数,是历史上最常用的激活函数,这一类激活函数以sigmoid函数和tanh函数为代表，如图8.22和图8.23所示。

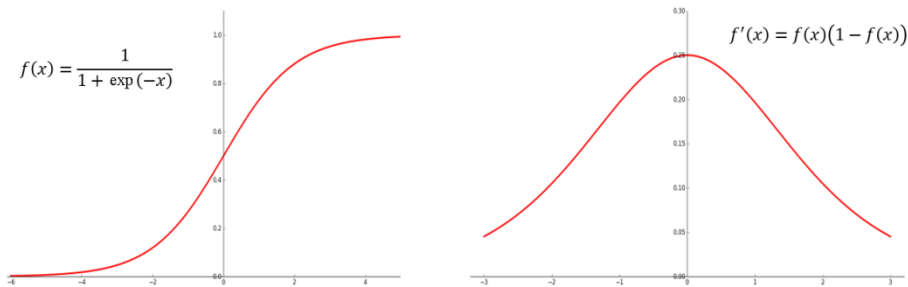


图8.22 sigmoid函数及其导数，sigmoid求导后，其导数的取值范围为 $[0, 1/4]$

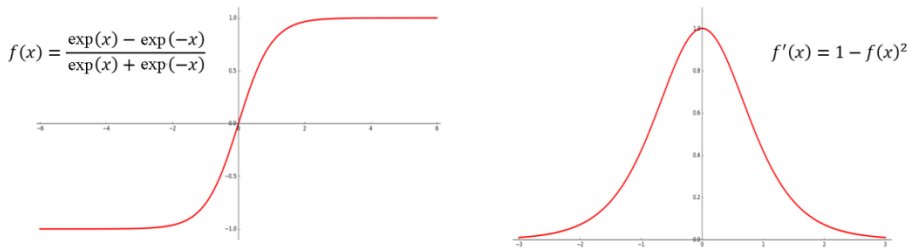


图8.23 tanh函数及其导数，tanh求导后，其导数的取值范围为 $[0,1]$



sigmoid函数把输入数据压缩到[0,1]范围内，处在中间部分的数据变化大，重要的特征会集中在这一部分区域，称为活跃区；相反，处在两侧的数据变化较小，神经元处在抑制的状态。sigmoid函数的表达形式为：

$$f(x) = \frac{1}{1 + e^{-x}} \quad (8.16)$$

近些年来sigmoid函数已经越来越少被采用，原因是它有下面两个不足。

第一，利用反向传播来训练神经网络时，sigmoid函数会产生梯度消失问题，导致训练深层网络的效果不理想。我们将在下一章详细讲解这个问题。

第二，经过sigmoid函数处理后的输出数据是一个非负值。假设第 $k$ 层隐藏层的输出数据向量为 $h^{k-1}(x) = (h^{k-1}(x)_1, h^{k-1}(x)_2, \dots, h^{k-1}(x)_{n_{k-1}})^T$ ，也就是说，经过sigmoid函数处理后，每一个神经元的输出数据 $h^{k-1}(x)_i$ 均大于0，它们的值将传递给下一层隐藏层，作为预激活函数 $a^k(x)$ 的输入，即有：

$$a^k(x) = b^k + W^k h^{k-1}(x), \quad k = 1, 2, \dots, L \quad (8.17)$$

利用式(8.17)对参数 $W^k$ 的求导有：

$$\frac{\partial a^k(x)}{\partial W^k} = h^{k-1}(x) > 0 \quad (8.18)$$

虽然在反向传播过程中， $W^k$ 的梯度并不完全由 $\frac{\partial a^k(x)}{\partial W^k}$ 的值决定，但这种不平衡性仍然会增加梯度的不稳定性。

另一个常用的激活函数系tanh函数，如图8.23所示，它把数值区间压缩到[-1,1]的范围内，可以把它看成是对sigmoid函数进行了按比例伸缩，事实上，它们两者的关系：

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1 \quad (8.19)$$

与sigmoid函数相比，tanh函数具有更稳定的梯度，这是因为经过tanh激活函数的处理后输出数据的均值约为0，相当于做了归一化的工作，避免了上面提到的sigmoid函数的第二个不足。此外，tanh函数的导数区间为[0,1]，比sigmoid函数的导数区间要大，在反向传播的过程中，衰减速度要比sigmoid函数慢。但由于tanh函数的导数小于1，因此利用反向传播来最优化神经网络模型时，同样无法避免梯度消失问题。

### 3. 近似生物神经元的激活函数

近年来,近似生物神经元的激活函数在神经网络中变得越来越流行,并且被证明在绝大多数网络和应用中,要比传统的sigmoid激活函数要好,这一类典型的激活函数以ReLU和softplus为代表。在2015年,它们被Yann LeCun、Yoshua Bengio和Geoffrey Hinton称为在深度学习领域最受欢迎的激活函数。

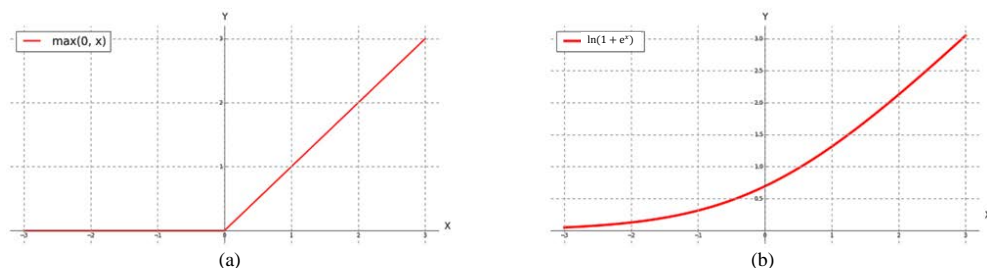


图8.24 (a)ReLU函数, (b)softplus函数, softplus函数可以看作是ReLU函数的平滑版本

ReLU全称为Rectified Linear Units, 其函数表达式及导数表达式分别为:

$$f(x) = \max(0, x) \quad (8.20)$$

$$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases} \quad (8.21)$$

与传统的sigmoid函数相比, ReLU激活函数具有下面的一些优点。

- 单侧抑制。观察式(8.21), 当输入小于0时, 神经元处于抑制态, 相反, 当输入大于0时, 神经元处于激活态。
- 相对宽阔的兴奋边界。考察图8.23和图8.24, 我们发现, 不管是sigmoid函数还是tanh函数, 它们的激活态都集中在中间的狭小范围内, 而ReLU则要宽阔很多, 只要输入大于0, 神经元都会处于激活态。
- 稀疏激活性<sup>[8]</sup>。相比其他激活函数, 稀疏性是ReLU的优势。sigmoid函数是tanh函数的导数, 我们发现这两个激活函数实际上是把抑制态的神经元置为一个非常小的值。非常小的值意味着它们仍然会参与到计算过程中, 但通过式(8.21), 我们发现ReLU直接把抑制态的神经元直接置为0, 使得这些神经元不再参与到后续的计算过程中, 这个细小的变化导致ReLU在实际的应用中, 收敛速度要远远快于其他的激活函数。

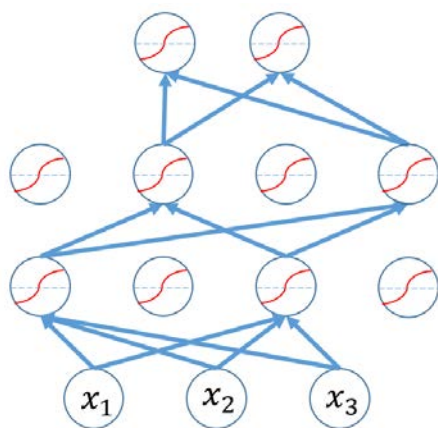


图8.25 ReLU函数使网络的训练产生一种稀疏连接关系，  
整个训练过程由激活态神经元来进行，而非激活态神经元则不参与计算

但ReLU的这种简单直接的处理方法也带来了一些缺点，最主要的缺点是导致神经网络的训练在后期变得脆弱，这时候，ReLU也被称为“dying ReLU”。产生这种现象的原因主要是ReLU对抑制态的处理过于极端，也就是将抑制态神经元的输出直接置为0，导致后续的训练中，抑制态神经元将不会再参与后面运算。

Leaky-ReLU是ReLU的改进版，也是为了解决“dying ReLU”问题而提出，它的函数形式为：

$$f(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases} \quad (8.22)$$

其中， $\alpha$ 是一个很小的常数，其函数图像如图8.26所示，我们可以发现Leaky-ReLU的最主要改变是在抑制侧，把原来的直接置为0重新置为一个很小的实数 $\alpha$ ，这个改动能有效缓解稀疏性导致的训练脆弱问题。

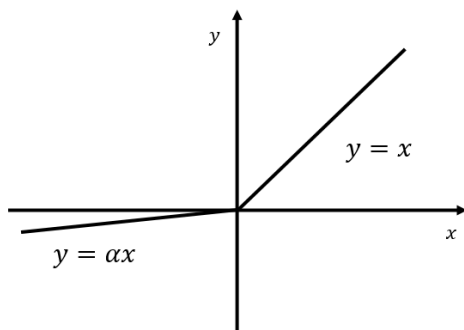


图8.26 Leaky-ReLU函数

在此基础上，研究人员还发明出了Parametric-ReLU和Randomized-Leaky这两个进化版本，它们是在Leaky-ReLU的基础上对 $\alpha$ 的取值进行改进。在Leaky-ReLU中， $\alpha$ 的取值是通过先验知识人工赋值的，而Parametric-ReLU则是把 $\alpha$ 的值作为一个参数，与神经网络的权重参数一起由反向传播算法求得。Randomized-Leaky是Leaky-ReLU的随机化版本，在训练过程中， $\alpha$ 是从一个高斯分布中随机取值，然后在测试过程中进行修正。更详细的细节可以参考文献[12]。

## 参考文献：

---

- [1] LeCun, Yann; Bengio, Yoshua; Hinton, Geoffrey (2015). "Deep learning". *Nature*. 521: 436–444.
- [2] McCulloch, W. S. and Pitts, W. H. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133.
- [3] Kurt Hornik (1991). Approximation Capabilities of Multilayer Feedforward Networks, *Neural Networks*, 4(2), 251–257. doi:10.1016/0893-6080(91)90009-T.
- [4] Haykin, Simon (1998). *Neural Networks: A Comprehensive Foundation*, Volume 2, Prentice Hall. ISBN 0-13-273350-1.
- [5] Hassoun, M. (1995). *Fundamentals of Artificial Neural Networks* MIT Press, p. 48.
- [6] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, Yoshua Bengio. Maxout Networks. *JMLR WCP* 28 (3): 1319–1327, 2013.
- [7] Rosenblatt, Frank (1957), The Perceptron--a perceiving and recognizing automaton. Report 85-460-1, Cornell Aeronautical Laboratory.
- [8] Xavier Glorot, Antoine Bordes, Yoshua Bengio. Deep Sparse Rectifier Neural Networks. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011. Pages: 315–323.
- [9] Andrew L. Maas, Awni Y. Hannun, Andrew Y. Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. *Proceedings of the 30th International Conference on Machine Learning*. 2013.
- [10] Andrej Karpathy, Justin Johnson. CS231n: Convolutional Neural Networks for Visual Recognition.
- [11] Yann LeCun, Leon Botton, Genevieve B. Orr, Klaus-Robert Miller. Efficient BackProp. *Neural Networks: tricks of the trade*, Springer, 1998.
- [12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *Computer Vision and Pattern Recognition*. 2015.

## 9

# 反向传播与梯度消失

上一章提到，反向传播算法仍然是当前深度神经网络最受欢迎的模型最优化方法。本章将会给出完整的BP算法流程，并分析为什么深度学习的训练具有很大的困难和挑战性。BP算法由Rumelhart等人在1986年提出<sup>[1]</sup>，是一种以梯度下降为核心的神经网络最优化方法。

为了后面讨论的方便，我们先来约定一些定义和术语，本章讨论的多层神经网络模型如图9.1所示，由输入层 $x$ 、 $L$ 层隐藏层( $h^1, h^2, \dots, h^L$ )和输出层 $f(x)$ 构成。

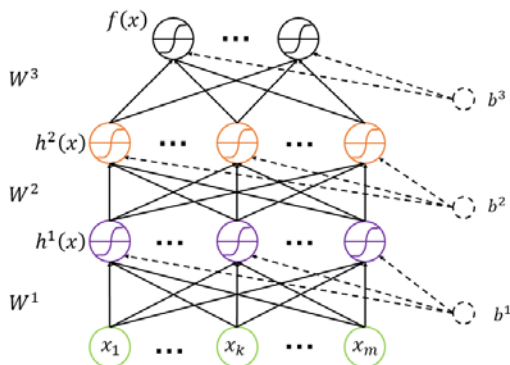


图9.1 神经网络模型

- 输入层记为 $x$ ，共有 $m$ 个神经元，记为 $x = (x_1, x_2, \dots, x_m)^T$ ，通常也把输入层记为第0层的隐藏层，即有：

$$x = h^0(x) \quad (9.1)$$

- 对于第 $k$ 层的隐藏层 $h^k(x)$ ,  $k = 1, 2, \dots, L$ , 由 $n_k$ 个神经元组成。隐藏层由预激活和激活两个阶段构成, 分别对应于预激活输出, 记为:

$$\begin{aligned} a^k(x) &= (a^k(x)_1, a^k(x)_2, \dots, a^k(x)_{n_k})^T \\ a^k(x) &= b^k + W^k h^{k-1}(x), \quad k = 1, 2, \dots, L \end{aligned} \quad (9.2)$$

和激活输出, 记为

$$\begin{aligned} h^k(x) &= (h^k(x)_1, h^k(x)_2, \dots, h^k(x)_{n_k})^T \\ h^k(x) &= g(a^k(x)), \quad k = 1, 2, \dots, L \end{aligned} \quad (9.3)$$

- $W^k$ 为连接第 $(k-1)$ 层的隐藏层中所有神经元和第 $k$ 层的隐藏层中所有神经元的权重参数矩阵, 大小为 $(n_{k-1} \times n_k)$ 。 $w_{ij}^k$ 表示连接第 $(k-1)$ 层的隐藏层中第 $i$ 个神经元和第 $k$ 层的隐藏层中第 $j$ 个神经元的权重。 $W_{i,*}^k = (w_{i1}^k, w_{i2}^k, \dots, w_{in_k}^k)^T$ 表示第 $(k-1)$ 层的隐藏层中第 $i$ 个神经元与第 $k$ 层的隐藏层中所有神经元的权重向量。
- $b^k$ 表示第 $k$ 层的隐藏层中所有神经元的偏移量, 记 $b^k = (b_1^k, b_2^k, \dots, b_{n_k}^k)^T$ , 其中 $b_j^k$ 表示第 $k$ 层隐藏层中的第 $j$ 个神经元的偏移量。
- 输出层记为 $f(x) = (f(x)_0, f(x)_1, \dots, f(x)_{c-1})^T$ , 其中 $f(x)_i$ 表示对于输入数据 $x$ , 预测为第 $i$ 类的概率。与隐藏层一样, 输出层同样由两个阶段构成, 分别对应于预激活输出, 记为:

$$a^{L+1}(x) = b^{L+1} + W^{L+1} h^L(x) \quad (9.4)$$

和激活输出, 记为:

$$f(x)_c = p(y = c|x) = \text{softmax}(a^{L+1}(x)) \quad (9.5)$$

与输入层表示一样, 为了使网络的表示方便统一, 常把输出层记为第 $L+1$ 层的隐藏层。

## 9.1 经验风险最小化

统计机器学习算法由模型、策略和算法三个要素构成，当选择了一种算法模型后，下一步需要考虑的是使用什么样的策略或准则来选择最优模型。

损失函数是机器学习中用于衡量模型一次预测结果好坏的函数，它是一个非负实数值函数，用 $L(Y, f(X))$ 来表示，常用的损失函数包括4种。

(1) 0-1损失函数。0-1损失函数比较的是预测值 $f(X)$ 与真实值 $Y$ 是否相同：

$$L(Y, f(X)) = \begin{cases} 1, & Y \neq f(X) \\ 0, & Y = f(X) \end{cases} \quad (9.6)$$

0-1损失函数是一个非凸函数，在求解的过程中，存在很多的不足，而且它只关心预测值与真实值是否相同，没有考虑预测值与真实值之间的距离，因此在实际应用中，0-1损失函数通常是作为一个衡量指标，而不是最优化的目标函数。

(2) 平方损失函数。平方损失函数是线性回归模型常用的最优化目标函数，记为：

$$L(Y, f(X)) = (Y - f(X))^2 \quad (9.7)$$

(3) 对数损失函数。常用于分类模型的最优化目标函数，记为：

$$L(Y, f(X)) = -\ln P(Y|X) \quad (9.8)$$

(4) Hinge损失函数：有时也称为最大间隔目标函数，是SVM采用的最优化目标：

$$L(Y, f(X)) = \max(0, 1 - Y \times f(X)) \quad (9.9)$$

对于任意给定的损失函数 $L(Y, f(X))$ ，可以求得平均意义下的期望损失函数，期望损失函数也称为期望风险函数 (Expected risk function)，记为：

$$R_{\text{exp}}(f) = E(L(Y, f(X))) = \int L(y, f(x))P(x, y)dx dy \quad (9.10)$$

机器学习学习的目标就是使期望风险函数最小，但由于联合分布函数 $P(x, y)$ 是不知道的，因此在实际应用中，通常的优化目标是经验风险最小化 (Empirical risk function)。形式化地说，假设现有训练数据集：

$$T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

模型 $f(x)$ 关于训练数据集 $T$ 的经验风险函数为：

$$R_{\text{emp}}(f) = \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) \quad (9.11)$$

事实上，由第4章概率论的大数定理可知，当 $N$ 趋向于无穷大时，有：

$$\lim_{N \rightarrow \infty} R_{\text{emp}}(f) = R_{\text{exp}}(f) \quad (9.12)$$

成立。为了防止出现过拟合的情况，通常会引入正则项，这样经验风险函数就变为结构风险函数（Structural risk function），记为：

$$R_{\text{struct}}(f) = \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) + \lambda \Omega(\theta) \quad (9.13)$$

因此，神经网络的最优化目标就是将结构风险函数最小化，对于多元分类模型，损失函数一般采用对数损失，记为：

$$L(y_i, f(x_i)) = -\ln p(y_i | x_i) = -\ln f(x_i)_{y_i}$$

故式(9.13)又可以化简为：

$$\min_{W, b} -\frac{1}{N} \sum_{i=1}^N \ln f(x_i)_{y_i} + \lambda (\Omega(W, b)) \quad (9.14)$$

## 9.2 梯度计算

BP算法的核心是梯度下降，本节首先来探讨神经网络的梯度计算过程，对式(9.14)的参数求导，主要难点在于对经验风险函数的求导。首先把对数损失函数写出通用的向量形式，即：

$$L(f(x), y) = -\sum_i 1_{(y=i)} \ln f(x)_i = -\ln f(x)_y \quad (9.15)$$

其中， $1_{(y=i)}$ 称为indicator function，满足：

$$1_{(y=i)} = \begin{cases} 1, & y = i \\ 0, & y \neq i \end{cases} \quad (9.16)$$

### 9.2.1 输出层梯度

利用BP算法来更新参数是一个反向的过程，即首先从输出层出发，一直返回到输入层，本节首先来考察输出层的梯度计算。



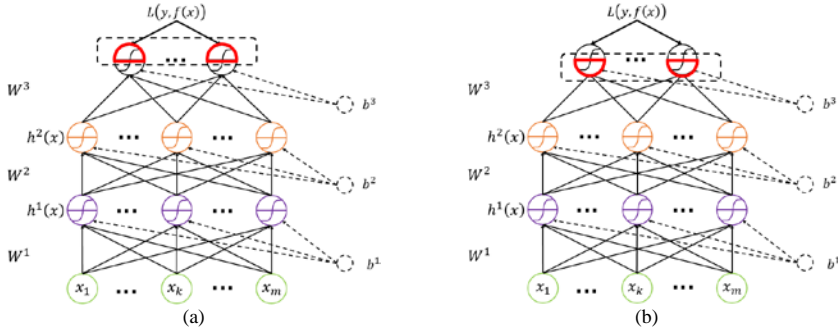


图9.2 (a)图为输出层的梯度, (b)图为输出层预激活梯度

输出层由预激活输出和激活输出两个阶段组成, 因此输出层的梯度也由激活输出梯度和预激活输出梯度两部分构成。

- 输出层激活输出偏导数:

$$\frac{\partial L(f(x), y)}{\partial f(x)_i} = \frac{\partial(-\ln f(x)_y)}{\partial f(x)_i} = \frac{-1_{(y=i)}}{f(x)_y} \quad (9.17)$$

- 输出层激活输出梯度:

$$\begin{aligned} \frac{\partial L(f(x), y)}{\partial f(x)} &= \frac{\partial(-\ln f(x)_y)}{\partial f(x)} \\ &= \frac{-1}{f(x)_y} (1_{(y=0)}, 1_{(y=1)}, \dots, 1_{(y=c-1)})^T \end{aligned} \quad (9.18)$$

记  $e(y) = (1_{(y=0)}, 1_{(y=1)}, \dots, 1_{(y=c-1)})^T$ , 式(9.18)可以化简为:

$$\frac{\partial L(f(x), y)}{\partial f(x)} = \frac{-e(y)}{f(x)_y} \quad (9.19)$$

- 输出层预激活输出偏导数:

$$\begin{aligned} \frac{\partial L(f(x), y)}{\partial a^{L+1}(x)_i} &= \frac{\partial(-\ln f(x)_y)}{\partial a^{L+1}(x)_i} = \frac{\partial(-\ln f(x)_y)}{\partial f(x)_y} \frac{\partial f(x)_y}{\partial a^{L+1}(x)_i} \\ &= \frac{-1}{f(x)_y} \frac{\partial f(x)_y}{\partial a^{L+1}(x)_i} \end{aligned} \quad (9.20)$$

其中  $f(x)_y$  满足:

$$f(x)_y = \text{softmax}(a^{L+1}(x)_y) = \frac{\exp(a^{L+1}(x)_y)}{\sum_{k=0}^{c-1} \exp(a^{L+1}(x)_k)} \quad (9.21)$$

把式(9.21)代入式(9.20)，利用链式求导法则，化简得：

$$\begin{aligned}
 \frac{\partial L(f(x), y)}{\partial a^{L+1}(x)_i} &= \frac{\partial(-\ln f(x)_y)}{\partial a^{L+1}(x)_i} = \frac{\partial(-\ln f(x)_y)}{\partial f(x)_y} \frac{\partial f(x)_y}{\partial a^{L+1}(x)_i} \\
 &= \frac{-1}{f(x)_y} \frac{\partial f(x)_y}{\partial a^{L+1}(x)_i} = \frac{-1}{f(x)_y} \left[ \frac{\partial}{\partial a^{L+1}(x)_i} \frac{\exp(a^{L+1}(x)_y)}{\sum_{k=0}^{c-1} \exp(a^{L+1}(x)_k)} \right] \\
 &= \frac{-1}{f(x)_y} \left[ \frac{\frac{\partial}{\partial a^{L+1}(x)_i} \exp(a^{L+1}(x)_y)}{\sum_{k=0}^{c-1} \exp(a^{L+1}(x)_k)} \right. \\
 &\quad \left. - \frac{\exp(a^{L+1}(x)_y) \times \left( \frac{\partial}{\partial a^{L+1}(x)_i} \sum_{k=0}^{c-1} \exp(a^{L+1}(x)_k) \right)}{(\sum_{k=0}^{c-1} \exp(a^{L+1}(x)_k))^2} \right] \\
 &= \frac{-1}{f(x)_y} \left[ \frac{1_{(y=i)} \exp(a^{L+1}(x)_y)}{\sum_{k=0}^{c-1} \exp(a^{L+1}(x)_k)} - \frac{\exp(a^{L+1}(x)_y)}{\sum_{k=0}^{c-1} \exp(a^{L+1}(x)_k)} \times \frac{\exp(a^{L+1}(x)_i)}{\sum_{k=0}^{c-1} \exp(a^{L+1}(x)_k)} \right] \\
 &= \frac{-1}{f(x)_y} [1_{(y=i)} f(x)_y - f(x)_y \times f(x)_i] \\
 &= -[1_{(y=i)} - f(x)_i]
 \end{aligned}$$

最终求得输出层预激活输出偏导数为：

$$\frac{\partial L(f(x), y)}{\partial a^{L+1}(x)_i} = -(1_{(y=i)} - f(x)_i) \quad (9.22)$$

• 输出层预激活输出梯度：

对式(9.22)进行扩展，得到输出层预激活梯度，下面不加推导直接给出结果：

$$\frac{\partial L(f(x), y)}{\partial a^{L+1}(x)} = -(e(y) - f(x)) \quad (9.23)$$

其中  $e(y) = (1_{(y=0)}, 1_{(y=1)}, \dots, 1_{(y=c-1)})^T$ ， $f(x) = (f(x)_0, f(x)_1, \dots, f(x)_{c-1})^T$ 。

### 9.2.2 隐藏层梯度

隐藏层是神经网络的核心层，对其的梯度求导也要比输出层复杂。与输出层一样，隐藏层同样由预激活输出和激活输出两个阶段组成，因此隐藏层的梯度也由激活输出梯度和预激活输出梯度两部分构成，为了使读者更好理解本节的推导过程，我们首先详细推导出第  $L$  层，即与输出层相连的最后一层隐藏层的梯度，然后不加推导和证明，

给出一般情况下的第 $k$ 层隐藏层的梯度公式。

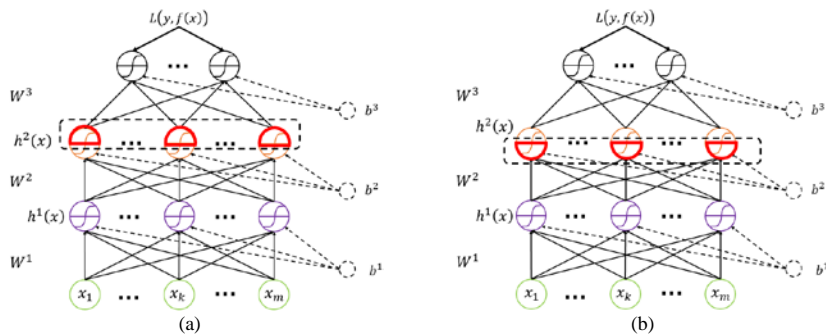


图9.3 (a)图为隐藏层的梯度, (b)图为隐藏层预激活梯度

• 隐藏层激活输出偏导数:

$$\frac{\partial L(f(x), y)}{\partial h^L(x)_i} = \frac{\partial -\ln f(x)_y}{\partial h^L(x)_i} \quad (9.24)$$

$h^L(x)_i$ 对应于第 $L$ 隐藏层的第 $i$ 个神经元的激活输出, 观察图9.4, 隐藏层的每一个神经元都与第 $L+1$ 层(即输出层)相关联, 且满足:

$$a^{L+1}(x)_j = \sum_{i=1}^{n_L} (w_{ij}^{L+1} \times h^L(x)_i) + b_j^{L+1} \quad (9.25)$$

因此利用链式法则, 有:

$$\frac{\partial L(f(x), y)}{\partial h^L(x)_i} = \sum_{j=0}^{c-1} \left( \frac{\partial L(f(x), y)}{\partial a^{L+1}(x)_j} \frac{\partial a^{L+1}(x)_j}{\partial h^L(x)_i} \right) \quad (9.26)$$

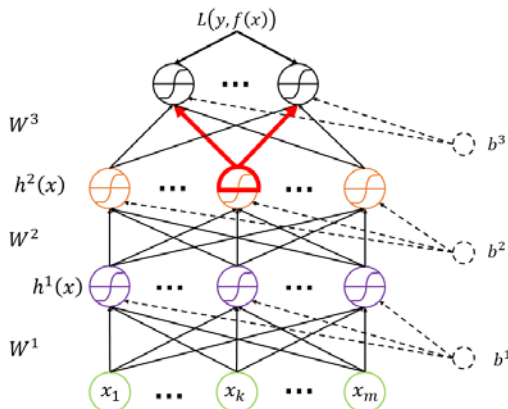


图9.4 神经网络的连接结构, 使得神经元的梯度可以通过链式法则来高效求解

把式(9.26)代入式(9.24)，化简得：

$$\frac{\partial L(f(x), y)}{\partial h^L(x)_i} = \frac{\partial -\ln f(x)_y}{\partial h^L(x)_i} = \sum_{j=0}^{c-1} \left( \frac{\partial -\ln f(x)_y}{\partial a^{L+1}(x)_j} \frac{\partial a^{L+1}(x)_j}{\partial h^L(x)_i} \right) \quad (9.27)$$

其中第一项已经在上一层的梯度计算中得到（对应于式(9.22)），即：

$$\frac{\partial -\ln f(x)_y}{\partial a^{L+1}(x)_j} = -(1_{(y=j)} - f(x)_j) \quad (9.28)$$

后一项可以通过对式(9.25)求导得：

$$\frac{\partial a^{L+1}(x)_j}{\partial h^L(x)_i} = w_{ij}^{L+1} \quad (9.29)$$

把式(9.29)和式(9.28)分别代入式(9.27)，得隐藏层偏导数：

$$\frac{\partial L(f(x), y)}{\partial h^L(x)_i} = (W_{i,*}^{L+1})^T \left( \frac{\partial -\ln f(x)_y}{\partial a^{L+1}(x)} \right) = -W_{i,*}^{L+1} (e(y) - f(x)) \quad (9.30)$$

- 隐藏层激活输出梯度：

只需要对式(9.30)做相应改变，就能很容易地给出隐藏层梯度公式，下面直接给出结果：

$$\frac{\partial L(f(x), y)}{\partial h^L(x)} = -W^{L+1} (e(y) - f(x)) \quad (9.31)$$

考察(9.31)式可以验证， $W^{L+1}$ 是一个大小为 $n_L \times c$ 的矩阵， $(e(y) - f(x))$ 是一个大小为 $c \times 1$ 的向量，故：

$$\frac{\partial L(f(x), y)}{\partial h^L(x)} = \left( \frac{\partial L(f(x), y)}{\partial h^L(x)_1}, \frac{\partial L(f(x), y)}{\partial h^L(x)_2}, \dots, \frac{\partial L(f(x), y)}{\partial h^L(x)_{n_L}} \right)^T \in \mathbf{R}^{n_L}$$

- 隐藏层预激活输出偏导数：

$$\frac{\partial L(f(x), y)}{\partial a^L(x)_i} = \frac{\partial -\ln f(x)_y}{\partial a^L(x)_i} = \frac{\partial -\ln f(x)_y}{\partial h^L(x)_i} \frac{\partial h^L(x)_i}{\partial a^L(x)_i} \quad (9.32)$$

其中第一项已经由式(9.30)求得，第二项可对式(9.31)求导，有：

$$\frac{\partial h^L(x)_i}{\partial a^L(x)_i} = g'(a^L(x)_i) \quad (9.33)$$

把式(9.33)和式(9.30)代入式(9.32)可得：

$$\frac{\partial L(f(x), y)}{\partial a^L(x)_i} = -W_{i,*}^{L+1}(e(y) - f(x))g'(a^L(x)_i) \quad (9.34)$$

• 隐藏层预激活输出梯度:

$$\frac{\partial L(f(x), y)}{\partial a^L(x)} = \frac{\partial L(f(x), y)}{\partial h^L(x)} \frac{\partial h^L(x)}{\partial a^L(x)} \quad (9.35)$$

把式(9.31)以及式(9.33)代入式(9.35), 化简得:

$$\begin{aligned} & \frac{\partial L(f(x), y)}{\partial a^L(x)} \\ &= \left( -W^{L+1}(e(y) - f(x)) \right) \times \left( g'(a^L(x)_1), g'(a^L(x)_2), \dots, g'(a^L(x)_{n_L}) \right) \end{aligned} \quad (9.36)$$

其中“ $\times$ ”表示element-wise product。下面给出在任意第 $k$ 隐藏层对应的梯度, 由于BP算法的过程是从输出层开始进行反向推导, 因此当求取第 $k$ 隐藏层的梯度时, 事实上已经得到了下面的结果。

$$\text{第 } t \text{ 隐藏层激活输出偏导数: } \frac{\partial L(f(x), y)}{\partial h^t(x)_i}, \quad t = k+1, \dots, L, L+1; \quad i = 1, 2, \dots, n_t$$

$$\text{第 } t \text{ 隐藏层激活输出梯度: } \frac{\partial L(f(x), y)}{\partial h^t(x)}, \quad t = k+1, \dots, L, L+1$$

$$\text{第 } t \text{ 隐藏层预激活输出偏导数: } \frac{\partial L(f(x), y)}{\partial a^t(x)_i}, \quad t = k+1, \dots, L, L+1; \quad i = 1, 2, \dots, n_t$$

$$\text{第 } t \text{ 隐藏层预激活输出梯度: } \frac{\partial L(f(x), y)}{\partial a^t(x)}, \quad t = k+1, \dots, L, L+1$$

由式(9.30)、式(9.31)、式(9.34)和式(9.36)的推导, 下面不加证明直接给出第 $k$ 隐藏层对应的结果。

$$\text{第 } k \text{ 隐藏层激活输出偏导数: } \frac{\partial L(f(x), y)}{\partial h^k(x)_i} = W_{i,*}^{k+1} \left( \frac{\partial -\ln f(x)_y}{\partial a^{k+1}(x)} \right) \quad (9.37)$$

$$\text{第 } k \text{ 隐藏层激活输出梯度: } \frac{\partial L(f(x), y)}{\partial h^k(x)} = W^{k+1} \left( \frac{\partial -\ln f(x)_y}{\partial a^{k+1}(x)} \right) \quad (9.38)$$

$$\text{第 } k \text{ 隐藏层预激活输出偏导数: } \frac{\partial L(f(x), y)}{\partial a^k(x)_i} = \frac{\partial -\ln f(x)_y}{\partial h^k(x)_i} g'(a^k(x)_i) \quad (9.39)$$

$$\begin{aligned} \text{第 } k \text{ 隐藏层预激活输出梯度: } & \frac{\partial L(f(x), y)}{\partial a^k(x)} = \frac{\partial -\ln f(x)_y}{\partial a^k(x)} \\ &= \frac{\partial L(f(x), y)}{\partial h^k(x)} \times \left( g'(a^k(x)_1), g'(a^k(x)_2), \dots, g'(a^k(x)_{n_k}) \right) \end{aligned} \quad (9.40)$$

### 9.2.3 参数梯度

上一节给出了输出层和隐藏层梯度计算的详细推导过程，现在可以开始计算神经网络的参数梯度。神经网络的参数，如图9.5所示。由连接相邻两层的权重参数 $W$ 和每一个神经元的偏移量 $b$ 组成，其中：

$$W = (W^1, W^2, \dots, W^{L+1})$$

$$b = (b^1, b^2, \dots, b^{L+1})$$

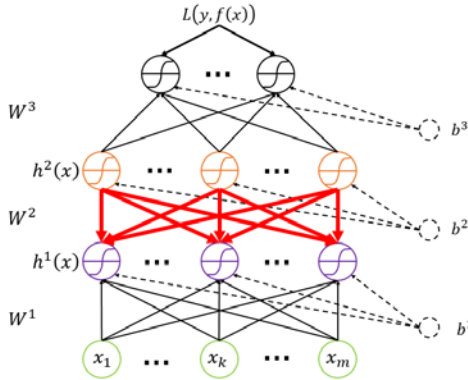


图9.5 神经网络的参数主要由两部分构成，分别是权重参数 $W$ 和神经元偏移量 $b$

对于任意的 $w_{i,j}^k$ ，有：

$$\frac{\partial L(f(x), y)}{\partial w_{i,j}^k} = \frac{\partial -\ln f(x)_y}{\partial w_{i,j}^k} = \frac{\partial -\ln f(x)_y}{\partial a^k(x)_j} \frac{\partial a^k(x)_j}{\partial w_{i,j}^k} \quad (9.41)$$

其中：

$$a^k(x)_j = b_j^k + \sum_i w_{i,j}^k h^{k-1}(x)_i \quad (9.42)$$

故有：

$$\frac{\partial a^k(x)_j}{\partial w_{i,j}^k} = h^{k-1}(x)_i \quad (9.43)$$

成立，把式(9.43)代入式(9.41)可得：

$$\frac{\partial L(f(x), y)}{\partial w_{i,j}^k} = \frac{\partial -\ln f(x)_y}{\partial w_{i,j}^k} = \frac{\partial -\ln f(x)_y}{\partial a^k(x)_j} h^{k-1}(x)_i \quad (9.44)$$

如果把式(9.44)写成矩阵的形式，对于任意的 $W^k$ ，有：

$$\frac{\partial L(f(x), y)}{\partial W^k} = \frac{\partial -\ln f(x)_y}{\partial W^k} = h^{k-1}(x) \left( \frac{\partial -\ln f(x)_y}{\partial a^k(x)} \right)^T \quad (9.45)$$

其中:

$$h^{k-1}(x) = (h^{k-1}(x)_1, h^{k-1}(x)_2, \dots, h^{k-1}(x)_{n_{k-1}})^T$$

我们考察式(9.45),  $h^{k-1}(x) \in \mathbf{R}^{n_{k-1} \times 1}$ ,  $\frac{\partial -\ln f(x)_y}{\partial a^k(x)} \in \mathbf{R}^{n_k \times 1}$ , 故有  $\frac{\partial L(f(x), y)}{\partial W^k} \in \mathbf{R}^{n_{k-1} \times n_k}$ , 且满足:

$$\left( \frac{\partial L(f(x), y)}{\partial W^k} \right)_{ij} = \frac{\partial L(f(x), y)}{\partial w_{i,j}^k}$$

同理, 对于任意的偏移量参数  $b_j^k$ , 有:

$$\frac{\partial L(f(x), y)}{\partial b_j^k} = \frac{\partial -\ln f(x)_y}{\partial b_j^k} = \frac{\partial -\ln f(x)_y}{\partial a^k(x)_j} \frac{\partial a^k(x)_j}{\partial b_j^k} = \frac{\partial -\ln f(x)_y}{\partial a^k(x)_j} \quad (9.46)$$

其中利用式(9.42)对  $b_j^k$  求导, 满足:

$$\frac{\partial a^k(x)_j}{\partial b_j^k} = 1$$

如果把式(9.46)写成矩阵的形式, 对于任意的  $b^k$ , 有:

$$\frac{\partial L(f(x), y)}{\partial b^k} = \frac{\partial -\ln f(x)_y}{\partial b^k} = \frac{\partial -\ln f(x)_y}{\partial a^k(x)} \quad (9.47)$$

至此, BP算法所需要的参数梯度全部推导完毕, 下面将给出完整的BP算法流程。

### 9.3 反向传播

反向传播由前向和后向两个操作构成, 如图9.6所示, 前向操作利用当前的权重参数和输入数据, 从下往上(即从输入层到输出层), 求取预测结果, 并利用预测结果与真实值求解出损失函数的值。反向操作则利用前向操作求解得到的损失函数, 从上往下(从输出层到输入层), 求解网络的参数梯度。经过前向和反向两个操作后, 完成了一次迭代过程。

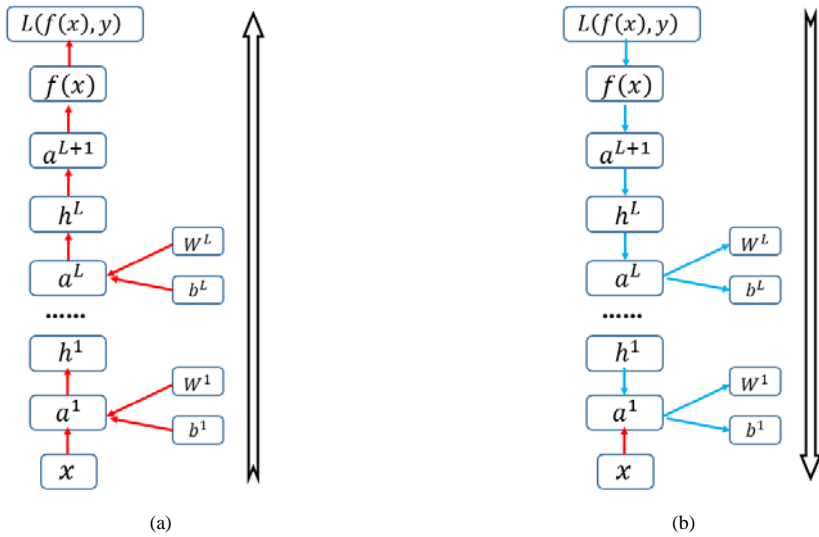


图9.6 (a)图：前向传播，从输入层开始，根据当前的权重参数信息，沿着网络前进，到达输出层，得到预测输出，把预测输出和真实值代入风险函数，求出当前的损失。(b)图：由上一步得到的损失，从输出层开始，反向求取每一层对应的参数梯度，利用梯度下降法更新参数

反向传播算法流程如图9.7所示。

#### 算法9.1 反向传播算法流程

$$1. \text{ 求取输出层的梯度 } \frac{\partial L(f(x), y)}{\partial a^{L+1}(x)} = -(e(y) - f(x)) \quad (9.23)$$

2. *for*  $k = L + 1, L, \dots, 1$  *do*

    计算参数梯度：

$$\frac{\partial L(f(x), y)}{\partial W^k} = \frac{\partial -\ln f(x)_y}{\partial W^k} = h^{k-1}(x) \left( \frac{\partial -\ln f(x)_y}{\partial a^k(x)} \right)^T \quad (9.45)$$

$$\frac{\partial L(f(x), y)}{\partial b^k} = \frac{\partial -\ln f(x)_y}{\partial b^k} = \frac{\partial -\ln f(x)_y}{\partial a^k(x)} \quad (9.47)$$

    计算第  $k-1$  隐藏层的激活和预激活梯度

$$\frac{\partial L(f(x), y)}{\partial h^k(x)} = W^{k+1} \left( \frac{\partial -\ln f(x)_y}{\partial a^{k+1}(x)} \right) \quad (9.38)$$

$$\frac{\partial L(f(x), y)}{\partial a^k(x)} = \frac{\partial L(f(x), y)}{\partial h^k(x)} \times (g'(a^k(x)_1), g'(a^k(x)_2), \dots, g'(a^k(x)_{n_k})) \quad (9.40)$$

图9.7 反向传播算法流程



## 9.4 深度学习训练的难点

随着深度学习被广泛应用于生活中的各个领域，伴随着各种突破性进展的同时，深度学习面临的各种困难也成为当前的研究热点，本节来对几个常见的难点问题介绍。

### 9.4.1 欠拟合——梯度消失

梯度消失（vanishing gradient），也称为梯度弥散，仍然是深度神经网络训练过程中所面临的最大挑战之一。梯度消失是如何产生的呢？梯度消失产生的源头就在于激活函数。回顾前面参数的求导过程，考察式(9.39)和式(9.40)，对预激活输出求导都牵扯到激活函数的导数，传统的激活函数及其导数，如sigmoid函数和tanh函数，如图9.8和图9.9所示。

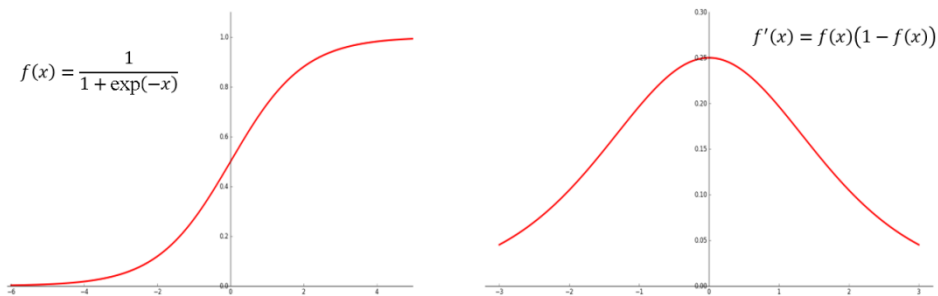


图9.8 sigmoid函数及其导数，sigmoid求导后，其导数的取值范围为 $[0, 1/4]$

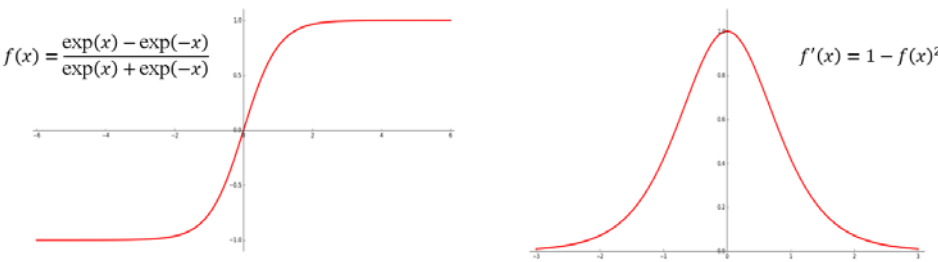


图9.9 tanh函数及其导数，tanh求导后，其导数的取值范围为 $[0,1]$

它们的导数取值范围都小于1。以sigmoid函数为例，sigmoid函数的取值范围为 $[0, 1]$ ，也就是说，当对激活函数求导后， $\frac{\partial L(f(x), y)}{\partial a^k(x)_i}$ 的取值都要比上一层减少 $\frac{1}{4}$ ，可以想象，梯度的计算是随着层数的增加而呈现指数级的递减趋势，离输出层越远，梯度减

少越明显，例如当有3个隐藏层时，到达 $h^1$ 时，它比 $h^3$ 要减少至少 $\left(\frac{1}{4}\right)^3 = \frac{1}{64}$ ，如图9.10所示。

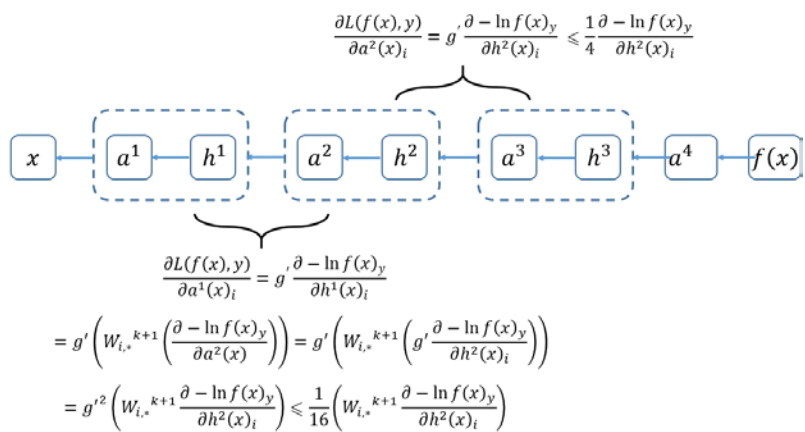


图9.10 梯度沿着反方向逐步递减

以Minist数据集为例，假设网络模型含有3个隐藏层，输入层到输出层的神经元个数分别为784、50、50、50、10。为了观察梯度消失的效果，可以考察参数梯度的变化，参数梯度向量的模可以反映出该参数学习速度的快慢， $\|\theta\|_2$ 越大，学习速度越快；反之，学习速度越慢，如图9.11所示。

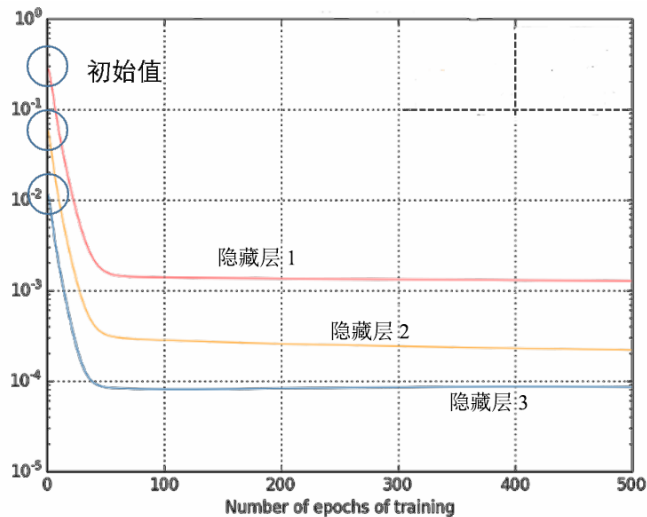


图9.11 观察3个隐藏层的梯度初始值和梯度变化，越靠近输出层的隐藏层它们的梯度值越大，也就是梯度的变化越明显，且越靠近输出层的隐藏层收敛更快

如何防止深度神经网络训练的梯度消失问题，或者说如何提升深度网络的训练效果，是当前深度学习一个非常热点的研究领域，并且取得了很多很好的效果，下面来考察当前几种常用的技巧。

- 采用更合理的激活函数，如ReLU、maxout<sup>[2]</sup>来取代传统的sigmoid函数系激活函数，这在一定程度上缓解了梯度消失的问题。
- **Batch Normalization**：是Google在2015年<sup>[5]</sup>提出的一种改进SGD算法的优化策略。BN由ZCA标准化层和重参数化层构成，如图9.12所示。

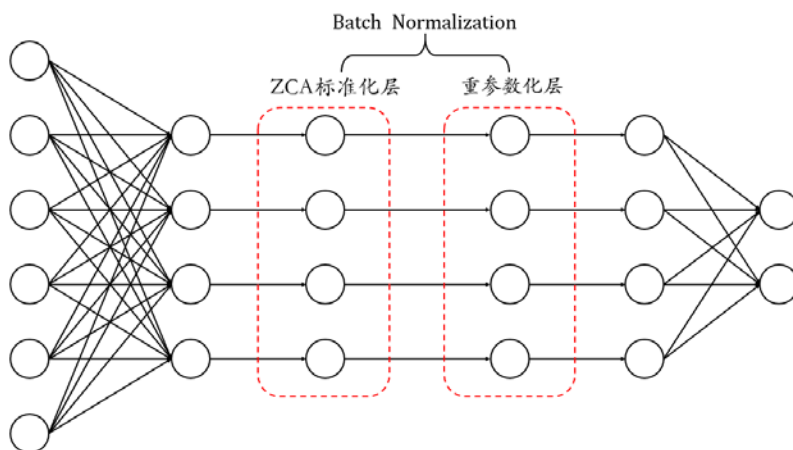


图9.12 Batch Normalization效果图

ZCA标准化层是标准化预激活输出，使得输出的每一个维度都服从标准正态分布的形式，即均值为0，方差为1。比如当前的预激活输出为 $f(w, x)$ ，则ZCA标准化的计算公式为：

$$f^*(w, x) = \frac{f(w, x) - \mu}{\sqrt{\sigma^2 + \epsilon}} \quad (9.48)$$

重参数化层主要是对标准化后的预激活输出结果执行scale和shift操作，如式(9.49)所示：

$$y(w, x) = \alpha f^*(w, x) + \beta \quad (9.49)$$

重参数化使得新的输出值有任意的均值和方差，这样梯度消失和非线性激活不再是一组矛盾关系，模型能够实时根据训练数据来动态决定激活层的非线性表达能力。关于BN更详细的解析读者可以参考文献[5]。

- 深度残差网络（Deep Residual Network）：将深度残差网络与卷积神经网络相结合的网络模型也简称为ResNet。它是微软研究院在2015年<sup>[3,4]</sup>提出的一种新的训练模式，深度残差网络的深度惊人，从原来的数十层提升到152层，深度远远超过了之前的深度网络结构，针对小数据更是设计了1001层的网络结构。第8章讲解激活函数时，提到激活函数如果是线性函数，那么将没有任何意义，即使再深层的网络也是等价于单层感知机，但如果增加的层是近似的单位映射，或者增加了某些干扰因素，由多个这样的近似单位映射构成残差单元，这些单元的残差组合能够近似某个复杂的函数。具体应用在CNN网络时，残差单元是由多个卷积层级联的输出和原有输入元素间相加，再经过ReLU激活后得到，如图9.13所示。更详细的原理分析可参考文献[3]。

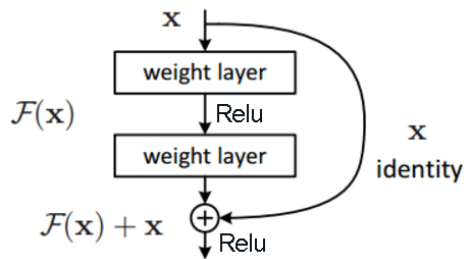


图9.13 残差单元（图片摘自参考文献[3]）

9.4.2 过拟合

过拟合（overfitting）是机器学习领域常见的难题。过拟合一般是指在模型选择中，选择的模型参数过多，导致对训练数据的预测很好，但对未知数据的预测很差的现象。神经网络，尤其是在深度神经网络领域，网络的层数更深，结构也更复杂，一般能达到数十层甚至上百层，而训练样本往往相对较少，过拟合的问题会更加严重。

正则化是机器学习中常用来解决过拟合的技巧。较为常见的正则化方法包括：对单模型，比如当验证集的效果变化不明显的时候可以提前终止迭代，或者采用L1正则化和L2正则化等。对多模型，可以利用boosting来集成提升，但在深度学习中，这种方法是不现实的，因为单个模型的训练已经非常复杂耗时，并且即使训练出多个网络模型，也难以在实际环境中做到快速集成。

Dropout结合了单模型和多模型的优点，在2014年由Hinton等人提出<sup>[6]</sup>，它是当前深度学习领域解决过拟合的强有力的武器。如果不考虑时间复杂度，可以通过训练

多个不同的网络模型来集成提升效果，网络结构之间差别越大，提升效果也会越明显，可以假设一个神经网络中有 $n$ 个不同神经元，那么相当于有 $2^n$ 个不同的网络结构，如图9.14所示。

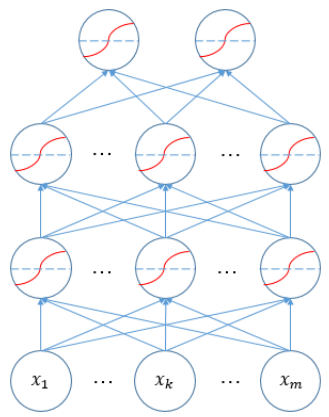


图9.14

Dropout的思想是每一迭代的过程中，会随机让网络某些节点（神经元）不参与训练，同时把与这些暂时丢弃的神经元（如图9.15所示的黑色结点）相关的所有边全部去掉，相应的权重不会在这一次迭代中更新，每一次迭代训练都重复这个操作。需要注意的是这些被丢弃的神经元只是暂时不做更新，下一次还是会重新参与随机化的Dropout。

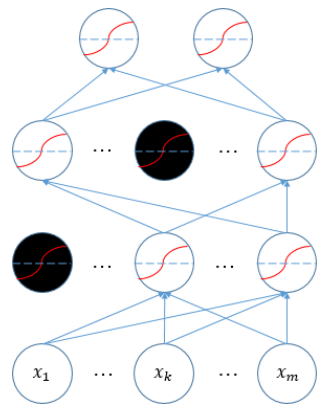


图9.15 Dropout效果图

Hinton给出了Dropout随机化选择的概率，对于隐藏层，一般取 $p = 0.5$ ，在大部分网络模型中，能达到或接近最优的效果，而对于输入层，被选中的概率要比被丢弃

的概率大，一般被选中的概率大约在 $p = 0.8$ 。在具体实现的时候，并没有对输入层做Dropout，在隐藏层取概率为0.5来进行正则化。

从理论角度来说，这个做法的巧妙之处在于，每一次的训练，网络都会丢弃部分的神经元，也就是相当于每一次迭代都在训练不同的网络结构，这样同一个模型多次Dropout迭代训练便达到了多模型融合的效果。

Hinton在其论文中还对Dropout给出了一种很有意思的生物学的解析，也就是为什么有性繁殖会比无性繁殖更能适应环境的变化，无性繁殖从母体中直接产生下一代，能保持母体的优良特性，不容易发生变异，但也因此造成了适应新环境能力差的缺点，而为了避免环境改变时物种可能面临的灭亡（相当于过拟合），有性繁殖除了会分别吸收父体和母体的基因之外，还会为了适应新环境而进行一定的基因变异。

## 参考文献：

---

- [1] Rumelhart, David E. Hinton, Geoffrey E.; Williams, Ronald J. (8 October 1986). "Learning representations by back-propagating errors". *Nature*. 323 (6088): 533–536.
- [2] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, Yoshua Bengio. Maxout Networks. *JMLR WCP* 28 (3): 1319–1327, 2013.
- [3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *CVPR* 2016.
- [4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity Mappings in Deep Residual Networks. Technical report, arXiv 2016.
- [5] Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ICML* 2015.
- [6] Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: A simple way to prevent neural networks from overfitting[J]. *The Journal of Machine Learning Research*, 2014, 15(1): 1929–1958.

## 自编码器及其相关模型

自编码器是一种无监督的神经网络模型，其核心的作用是能够学习到输入数据的深层表示。自编码器在早期有一个非常重要的应用，就是用来初始化神经网络的权重参数，实践证明，这种通过逐层预训练( Layer-wise pre-training )加微调( fine-tuning )得到的初始化参数要比传统的对称随机初始化参数效果好<sup>[1]</sup>，更容易收敛，并且在一定程度上缓解了BP算法在深层网络训练中出现的梯度消失问题。事实上，这种预训练的方法正是深度学习在2006年兴起的核心技术<sup>[2]</sup>。但后来，随着深度学习研究理论不断发展，有很多先进的技巧被提出，并且效果都比逐层预训练好，包括Batch Normalization、深度残差网络等，尤其是残差网络的提出使得训练任意深度的网络成为可能。

当前自编码器的主要应用有两个方面，一是特征提取；另一个是非线性降维，用于高维数据的可视化，与流形学习( Manifold Learning )关系密切。本章将探讨几种常见的自编码器及其变种网络，简要介绍它们的原理和应用。

### 10.1 自编码器

自编码器( AutoEncoder，简称AE )，是在20世纪80年代被提出的一种神经网络模型<sup>[3]</sup>，最原始的AE网络是一个三层的前馈神经网络结构，由输入层、隐藏层和输出层构成，如图10.1所示。

如图10.1所示, 首先对AE网络的结构与符号进行定义: 对输入层, 用小写字母 $x$ 来表示, 满足 $x = \{x_1, \dots, x_n\}$ ,  $n$ 表示输入层的神经元个数,  $x_i$ 表示输入层的第 $i$ 个神经元取值, 在图中用空心圆来表示。隐藏层用小写字母 $h$ 来表示, 满足 $h = \{h_1, \dots, h_m\}$ ,  $m$ 表示隐藏层的神经元个数,  $h_j$ 表示隐藏层的第 $j$ 个神经元取值, 在图中用实心圆来表示。输出层用小写字母 $\hat{x}$ 来表示, 其中 $\hat{x} = \{\hat{x}_1, \dots, \hat{x}_n\}$ , 输出层的神经元个数与输入层的神经元个数相同, 在图中用空心圆来表示。

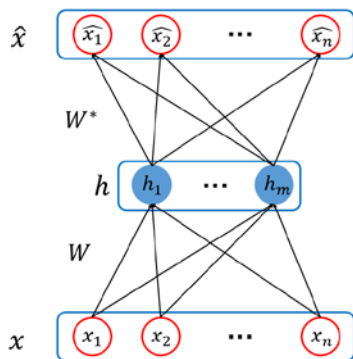


图10.1 自编码器结构图

整个AE由下面两个操作构成: 第一个操作是数据从输入层到隐藏层的过程, 我们称为编码 (Encoder), 其过程如式(10.1)所示:

$$h(x) = f(W^T x + b) \quad (10.1)$$

其中,  $f$ 是激活函数, 第二个操作是从隐藏层到输出层的过程, 称之为解码 (Decoder), 其计算过程如式(10.2)所示, 其中如果 $W^*$ 的取值满足 $W^T = W^*$ , 称为tied weights。

$$\hat{x} = f((W^*)^T h(x) + c) \quad (10.2)$$

AE是一个无监督学习的网络, 其学习目的是将输入层数据 $x$ 通过转换得到其隐藏层的表示 $h(x)$ , 然后由隐藏层重构, 还原出新的输入数据 $\hat{x}$ 。AE的训练目标就是使得重构后的数据 $\hat{x}$ 能够尽量还原输入层数据 $x$ 。由此, 得到构建AE的损失函数: 对于二值神经网络, 也就是输入层的每个神经元只能取值0或1, 那么损失函数通常由交叉熵来定义:

$$L(W, b, c) = - \sum_{k=1}^n (x_k \log \hat{x}_k + (1 - x_k) \log(1 - \hat{x}_k)) \quad (10.3)$$



若输入神经元是一个任意实数，则通常采用均方误差来定义损失函数：

$$L(W, b, c) = \frac{1}{2} \sum_{k=1}^n (x_k - \hat{x}_k)^2 \quad (10.4)$$

利用梯度下降等最优化算法，可以求解出模型的最优参数 $W$ 、 $b$ 和 $c$ 。最后，从上面的分析中可以发现自编码器的核心设计是隐藏层，隐藏层的设计有两种方式，如图10.2所示。

(1) 当隐藏层神经元个数小于输入层神经元个数时，称为undercomplete。该隐藏层设计使得输入层到隐藏层的变换本质上是一种降维的操作，网络试图以更小的维度去描述原始数据而尽量不损失数据信息，从而得到输入层的压缩表示。当隐藏层的激活函数采用线性函数时，自编码器也被称为线性自编码器，其效果等价于主成分分析（PCA）。

(2) 当隐藏层神经元个数大于输入层神经元个数时，称为overcomplete。该隐藏层设计一般用于稀疏编码器，可以获得稀疏的特征表示，也就是隐藏层中有大量的神经元取值为0。

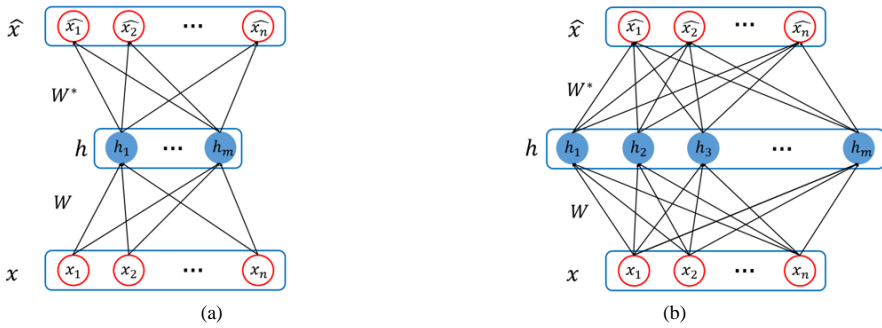


图10.2 自编码器隐藏层的两种设计形态，(a)图undercomplete，(b)图overcomplete

## 10.2 降噪自编码器

降噪自编码器（Denoising Autoencoder，简称DAE），由Bengio在2008年提出<sup>[4]</sup>，其目的是增强自编码器的鲁棒性。自编码器的目标是期望重构后的结果输出 $\hat{x}$ 与输入数据 $x$ 相同，也就是能够学习到输入层的正确数据分布。但当输入层数据受到噪音的影响时，可能会使获得的输入数据本身就不服从原始的分布。在这种情况下，利用自编码器得到的结果也将是不正确的，为了解决这种由于噪音产生的数据偏差问题，我

们提出了DAE的网络结构，如图10.3所示。与自编码器相比，DAE在输入层与隐藏层之间添加了噪音处理，得到新的经过处理后的噪音层数据为 $\tilde{x}$ ，然后按照这个新的噪音数据 $\tilde{x}$ 进行常规自编码器变换操作。

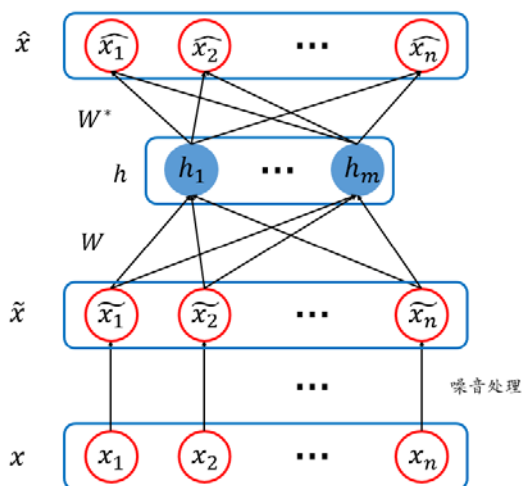
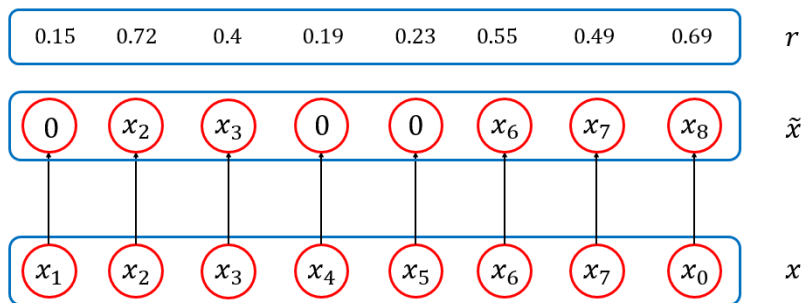


图10.3 降噪自编码器结构图

注意，DAE的损失函数与AE的损失函数是一样的，也就是说DAE的输出层仍然是最大限度地还原输入层 $x$ ，而不是经过噪音处理后的噪音层数据 $\tilde{x}$ ，则损失函数仍然满足：

$$L(W, b, c) = \begin{cases} -\sum_{k=1}^n (x_k \log \hat{x}_k + (1 - x_k) \log(1 - \hat{x}_k)) & x_k \text{ 是一进制} \\ \frac{1}{2} \sum_{k=1}^n (x_k - \hat{x}_k)^2 & \text{其他} \end{cases}$$

从输入层 $x$ 到噪音层 $\tilde{x}$ 的处理方式有很多，最常用的方式是mask noise。设置一个噪音阈值 $p$ ，满足 $0 < p < 1$ ，每一次生成一个大小从0到1的浮点数 $r$ ，当 $r < p$ 时，令 $\tilde{x}_i = 0$ ；否则令 $\tilde{x}_i = x_i$ ，现设 $p = 0.3$ ，则从输入层 $x$ 到噪音层 $\tilde{x}$ 的过程如图10.4所示。

图10.4 从输入层 $x$ 到噪音层 $\tilde{x}$ 的变化

其他构建  $\tilde{x}$  的方式还包括高斯噪音 ( Gaussian noise )，它满足  $\tilde{x} = x + \text{Normal}(0,1)$ 。DAE之所以具有更好的鲁棒性，我们可以通过下面的流形解析来理解<sup>[4]</sup>。假设原始数据 $x$ 的分布是如图10.5所示的流形曲线，经过噪音 $p(\tilde{x}|x)$ 处理后，得到的 $\tilde{x}$ 将偏离于流形曲线，然后通过三层的自编码器来重构数据 $x$ ，重构的过程相当于将 $\tilde{x}$ 重新投影回流形曲线上，由于 $\tilde{x}$ 与真实分布之间有偏差，从而增强了网络对数据噪音的容忍度。

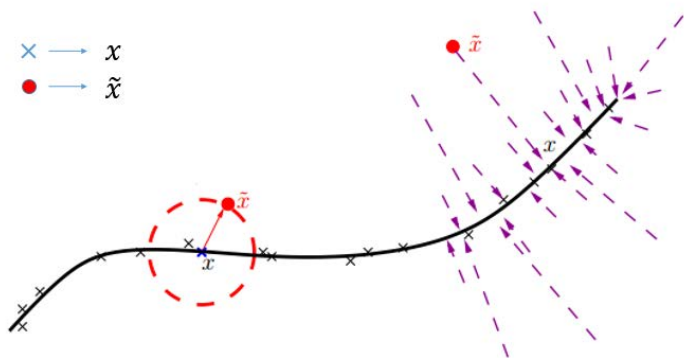


图10.5 在二维空间中的一维流形分布

### 10.3 栈式自编码器

栈式自编码器 ( Stacked Autoencoders, 简称SAE ), 也被称为堆栈自编码器、堆叠自编码器等。在第1章已经介绍了深度网络对输入数据的处理是一个分层的过程，即从原始数据开始，得到简单的边缘表示，然后在此基础上构建更复杂的结构表示。单个自编码器通过三层结构能够将输入层 $x$ 抽象为隐藏层表示 $h$ ，那么一种很自然的想

法是将多个自编码器进行叠加，利用上一层的隐藏层表示作为下一层的输入，得到更抽象的表示。具体来说，对第*i*个自编码器进行训练，得到其输入层数据 $x^i$ 的隐藏层表示 $h^i$ ，以及输出层 $\hat{x}^i$ 。在此基础上，构建第*i* + 1个自编码器，并丢弃第*i*个自编码器的输出数据 $\hat{x}^i$ ，以 $h^i$ 作为第*i* + 1个自编码器的输入数据，则 $x^{i+1} = h^i$ ，重复执行这个操作，将多个自编码器逐层叠加，得到如图10.6所示的SAE结构表示。

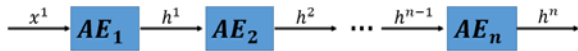


图10.6 栈式自编码器结构

SAE的一个很重要应用是通过逐层预训练来初始化网络权重参数，从而提升深层网络的收敛速度和减缓梯度消失的影响。对于常见的监督学习，SAE通过下面两个阶段作用于整个网络。

### 1. 逐层预训练

逐层预训练是指通过自编码器来训练每一层的参数，作为神经网络的初始化参数，以图10.7为例，图中包含一个输入层（ $x$ ）、两个隐藏层（ $h^1, h^2$ ）和输出层（ $o$ ），网络模型参数包括： $\{W^1, b^1, W^2, b^2, W^3, b^3\}$ 。

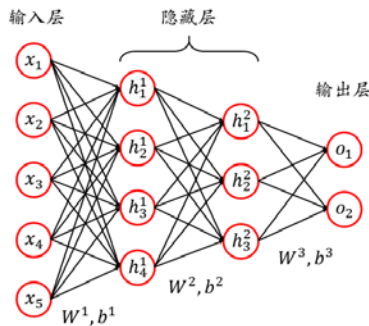


图10.7 一个4层的前馈神经网络

通常，神经网络隐藏层的权重参数初始化一般采用文献[1]提出的方案，即当激活函数采用tanh函数时， $W$ 的初始化随机值满足式(10.5)所示的均匀分布采样，其中 $input\_size$ 表示由 $W$ 连接的两个网络层中输入层的神经元个数， $output\_size$ 表示输出层的神经元个数。

$$W \sim \text{uniform}\left(-\sqrt{\frac{6}{input\_size + output\_size}}, \sqrt{\frac{6}{input\_size + output\_size}}\right) \quad (10.5)$$

当激活函数采用sigmoid函数时,  $W$ 的初始化随机值满足式(10.6)所示的均匀分布采样:

$$W \sim \text{uniform}\left(-4 \times \sqrt{\frac{6}{\text{input\_size} + \text{output\_size}}}, 4 \times \sqrt{\frac{6}{\text{input\_size} + \text{output\_size}}}\right) \quad (10.6)$$

利用逐层预训练的方法, 首先构建多个自编码器, 每一个自编码器对应于一个隐藏层, 对于图10.7所示的前馈网络, 可以构建如图10.8所示的两个自编码器。

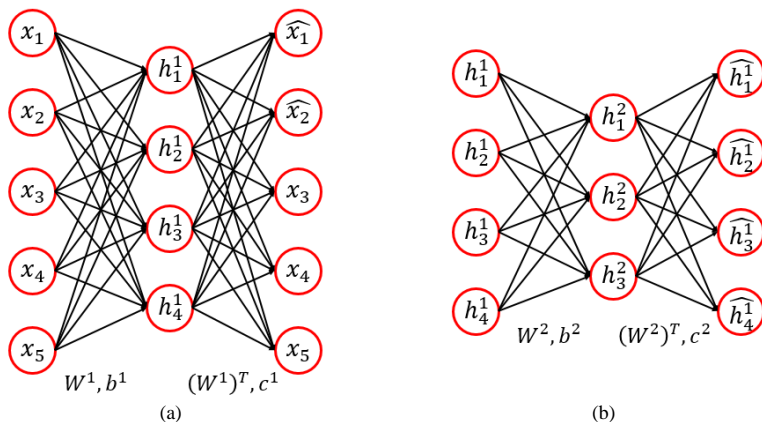


图10.8 为图10.7构建对应的两个自编码器, 用于预训练参数 $\{W^1, b^1, W^2, b^2\}$

其中, (a)图对应的是第一层隐藏层 $h^1$ 的自编码器, 输入层与前馈神经网络的输入层一致, 输出层是输入层的重构 $\hat{x}$ 。(b)图对应是第二层隐藏层 $h^2$ 的自编码器, 输入层是第一层隐藏层 $h^1$ 的值, 输出层是对 $h^1$ 的重构 $\hat{h}^1$ 。从左到右逐层训练每一个自编码器, 用训练后的最优参数作为神经网络的初始化参数。

如果考虑到模型的鲁棒性, 防止数据受噪音的影响, 可以将AE变为DAE, 这样由多个DAE叠加的栈式自编码器, 也称为栈式降噪自编码器 (Stacked Denoising Autoencoders, 简称SDAE)。

## 2. 微调

经过第一步的逐层预训练后, 得到了网络权重参数更加合理的初始化估算, 就可以像训练普通的深层网络一样, 通过输出层的损失函数, 利用梯度下降等方法来迭代求解最优参数。

## 10.4 稀疏编码器

本节介绍另外一种自编码器的变种网络：稀疏编码<sup>[5]</sup>（Sparse Coding）。由于稀疏编码器能够学习到输入数据的稀疏特征表示，因此当前被广泛应用于无监督的特征提取学习中。下面首先给出稀疏编码的数学定义：稀疏编码的网络结构与自编码器一样，同样是一个由三层结构构成的前馈神经网络，在稀疏编码中，期望模型能够对任意的输入数据 $x^t$ ，得到隐藏层表示 $h^t$ ，以及输出表示 $\widehat{x^t}$ 。并且 $x^t$ 、 $h^t$ 和 $\widehat{x^t}$ 满足下面的两个性质。

- （1）隐藏层向量是稀疏的，则向量 $h^t$ 有尽可能多的零元素。
- （2）输出层数据 $\widehat{x^t}$ 能够尽可能还原输入层数据 $x^t$ 。

设现有由 $T$ 个训练数据构建数据集 $\{x^1, x^2, \dots, x^T\}$ ，稀疏编码的目标是求取最优的矩阵 $D$ 和隐藏层 $h^t$ ，使得模型满足上面的两个性质，用数学表达式可以表示为：

$$\min_D \left( \frac{1}{T} \sum_{t=1}^T \min_{h^t} \left( \frac{1}{2} \|x^t - Dh^t\|_2^2 + \lambda \|h^t\|_1^2 \right) \right) \tag{10.7}$$

其中， $D$ 表示隐藏层到输出层的权重参数矩阵，满足 $Dh^t = \widehat{x^t}$ ，矩阵 $D$ 也被称为“字典”，大小为 $n \times m$ 维， $m$ 是隐藏层向量的大小， $n$ 是输出层数据的大小。如果将矩阵 $D$ 写成列向量组的表示形式，则满足：

$$D = (\phi_1, \phi_2, \dots, \phi_m)$$

$\phi_i$ 是 $n$ 维向量，刻画了从输入数据学到的不同特征表示，对应的隐藏层神经元 $h_i^t$ 描述了 $\phi_i$ 对数据 $x^t$ 的重要性贡献，它们之间的关系可以表示为：

$$x^t \approx \widehat{x^t} = \sum_{i=1}^m h_i^t \phi_i \tag{10.8}$$

其中 $h_i^t$ 有大量元素为0，如果得到如图10.9的结果，那么对应的隐藏层向量为：  
 $h^t = (0, 0.3, 0, \dots, 0, 0.5, 0, \dots, 0, 1, 0, \dots, 0, 0.8, 1, 0, \dots, 0.5, \dots)^T$ 。

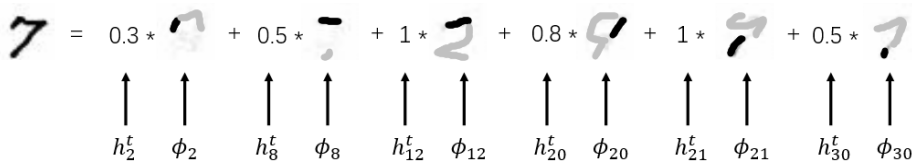


图10.9 输入数据的稀疏编码表示

式(10.7)由两个最小化构成, 首先单独考察每一个最小化的求解。设 $\mathbf{D}$ 已知, 隐藏层 $\mathbf{h}^t$ 由式(10.7)的内层最小化公式求得, 则满足:

$$\mathbf{h}^t = \operatorname{argmin}_{\mathbf{h}^t} \left( \frac{1}{2} \|\mathbf{x}^t - \mathbf{D}\mathbf{h}^t\|_2^2 + \lambda \|\mathbf{h}^t\|_1^2 \right) \quad (10.9)$$

考察式(10.9), 函数的最小化由两部分构成, 其中 $\min \frac{1}{2} \|\mathbf{x}^t - \mathbf{D}\mathbf{h}^t\|_2^2$ 与自编码器的原理一致, 是为了保证输出层数据 $\hat{\mathbf{x}}^t$ 能够尽可能还原输入层数据 $\mathbf{x}^t$ , 称为reconstruction error。 $\min \lambda \|\mathbf{h}^t\|_1^2$ 是为了能够让向量 $\mathbf{h}^t$ 有尽可能多的零元素, 称为sparsity penalty。两个函数的图像如图10.10所示。

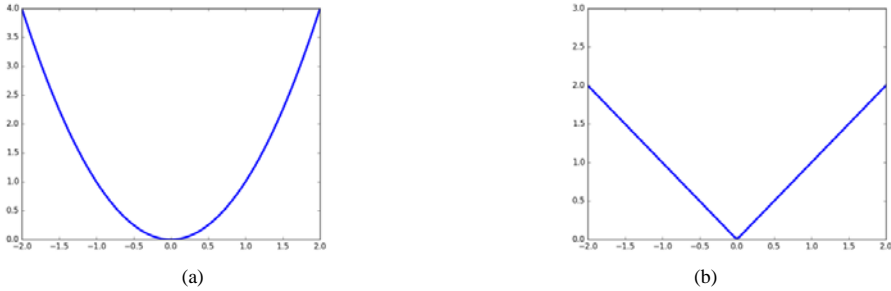


图10.10 (a)图reconstruction error的函数图像, (b)图sparsity penalty的函数图像

要最小化式(10.9), 可以利用梯度下降法来求解, 设损失函数为:

$$L(\mathbf{x}^t) = \frac{1}{2} \|\mathbf{x}^t - \mathbf{D}\mathbf{h}^t\|_2^2 + \lambda \|\mathbf{h}^t\|_1^2 \quad (10.10)$$

式(10.10)对 $\mathbf{h}^t$ 求导, 得:

$$\nabla_{\mathbf{h}^t} (L(\mathbf{x}^t)) = \mathbf{D}^T (\mathbf{D}\mathbf{h}^t - \mathbf{x}^t) + \lambda \times \operatorname{sign}(\mathbf{h}^t) \quad (10.11)$$

设 $\mathbf{h}^t = (h_1^t, h_2^t, \dots, h_m^t)^T$ , 对 $\mathbf{h}^t$ 的每一个隐藏层神经元 $h_k^t$ 单独求导, 有:

$$\frac{\partial}{\partial h_k^t} (L(\mathbf{x}^t)) = (\mathbf{D}_{*,k})^T (\mathbf{D}\mathbf{h}^t - \mathbf{x}^t) + \lambda \times \operatorname{sign}(h_k^t) \quad (10.12)$$

但由于1-范数在0处不可导, 解决方法是比较sparsity penalty的求导结果是否改变了 $h_k^t$ 的符号, 具体来说就是首先将reconstruction error的求导结果用于更新 $h_k^t$ , 则式(10.12)等式右边的第一项, 有:

$$h_k^t \Leftarrow h_k^t - \alpha \times (\mathbf{D}_{*,k})^T (\mathbf{D}\mathbf{h}^t - \mathbf{x}^t) \quad (10.13)$$

然后考察sparsity penalty的求导结果, 比较 $h_k^t - \alpha \times \lambda \times \operatorname{sign}(h_k^t)$ 与 $h_k^t$ 的符号, 若不相等, 则直接将 $h_k^t$ 的值设为0, 即:

$$\text{sign}(h_k^t - \alpha \times \lambda \times \text{sign}(h_k^t)) \neq \text{sign}(h_k^t) \Rightarrow h_k^t = 0 \quad (10.14)$$

否则，令：

$$h_k^t = h_k^t - \alpha \times \lambda \times \text{sign}(h_k^t) \quad (10.15)$$

将上面的处理方法称为**迭代阈值缩减算法**（Iterative Shrinkage Thresholding Algorithm，简称ISTA）， $\mathbf{h}^t$ 的迭代更新如算法如图 10.11所示。

---

算法10.1 ISTA

---

1. 初始化 $\mathbf{h}^t = (0, 0, \dots, 0)^T$
2. 当 $\mathbf{h}^t$ 没有收敛时，重复执行下面的更新操作：

$$\mathbf{h}^t \leftarrow \mathbf{h}^t - \alpha \times \mathbf{D}^T(\mathbf{D}\mathbf{h}^t - \mathbf{x}^t)$$

$$\mathbf{h}^t \leftarrow \text{shrink}(\mathbf{h}^t, \alpha \times \lambda)$$

3. 输出 $\mathbf{h}^t$
- 

图10.11 ISTA算法对 $\mathbf{h}^t$ 的迭代更新过程

其中，shrink函数的表达式如下所示。

$$\text{shrink}(a_i, b_i) = \text{sign}(a_i) \times \max(|a_i| - b_i, 0) \quad (10.16)$$

式(10.16)的结果实际上是式(10.14)和式(10.15)的组合，设 $b_i = 1$ ，则 $\text{shrink}(a_i, b_i)$ 的取值与 $a_i$ 的关系如图10.12所示。

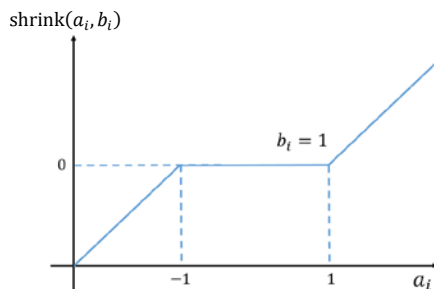


图10.12  $\text{shrink}(a_i, b_i)$ 的取值随着 $a_i$ 的变化图

接下来考察式(10.7)的外层最小化，当固定 $\mathbf{h}^t$ 时，应该如何求取字典 $\mathbf{D}$ 呢？ $\mathbf{D}$ 通过最优化下面的损失函数来得到：



$$\min_{\mathbf{D}} \left( \frac{1}{T} \sum_{t=1}^T L(\mathbf{x}^t) \right) = \min_{\mathbf{D}} \left( \frac{1}{T} \sum_{t=1}^T \min_{\mathbf{h}^t} \left( \frac{1}{2} \|\mathbf{x}^t - \mathbf{D}\mathbf{h}^t\|_2^2 + \lambda \|\mathbf{h}^t\|_1^2 \right) \right) \quad (10.17)$$

由于固定 $\mathbf{h}^t$ ，因此可以忽略1-范数，则重新整理后的目标函数为：

$$\min_{\mathbf{D}} \left( \frac{1}{T} \sum_{t=1}^T \min_{\mathbf{h}^t} \left( \frac{1}{2} \|\mathbf{x}^t - \mathbf{D}\mathbf{h}^t\|_2^2 \right) \right) \quad (10.18)$$

同样可以利用梯度下降法求解，将(10.18)式对 $\mathbf{D}$ 求导，满足：

$$\frac{\partial}{\partial \mathbf{D}} (L(\mathbf{D}, \mathbf{x}^t)) = \frac{1}{T} \sum_{t=1}^T (\mathbf{x}^t - \mathbf{D}\mathbf{h}^t)(\mathbf{h}^t)^T \quad (10.19)$$

这样对参数 $\mathbf{D}$ 的更新策略为：

$$\mathbf{D} \leftarrow \mathbf{D} + \alpha \times \frac{1}{T} \sum_{t=1}^T (\mathbf{x}^t - \mathbf{D}\mathbf{h}^t)(\mathbf{h}^t)^T \quad (10.20)$$

同时可以将 $\mathbf{D}$ 的列向量 $\mathbf{D}_{*,j}$ 进行规则化操作，满足：

$$\mathbf{D}_{*,j} = \frac{\mathbf{D}_{*,j}}{\|\mathbf{D}_{*,j}\|} \quad (10.21)$$

事实上，对字典 $\mathbf{D}$ 的求解除了可以利用梯度下降算法求解外，可以考察另一种最优化算法：block coordinate descent。利用极值存在的必要条件，将式(10.18)对 $\mathbf{D}$ 求导，并令导数为0，有：

$$\begin{aligned} \frac{\partial}{\partial \mathbf{D}_{*,j}} (L(\mathbf{D}, \mathbf{x}^t)) &= \frac{1}{T} \sum_{t=1}^T (\mathbf{x}^t - \mathbf{D}\mathbf{h}^t)h_j^t = 0 \\ \Leftrightarrow \sum_{t=1}^T \left( \mathbf{x}^t - \left( \sum_{i \neq j} \mathbf{D}_{*,i} h_i^t \right) - \mathbf{D}_{*,j} h_j^t \right) h_j^t &= 0 \\ \Leftrightarrow \sum_{t=1}^T \left( \mathbf{x}^t - \left( \sum_{i \neq j} \mathbf{D}_{*,i} h_i^t \right) \right) h_j^t &= \sum_{t=1}^T \mathbf{D}_{*,j} (h_j^t)^2 \\ \Leftrightarrow \mathbf{D}_{*,j} &= \frac{1}{\sum_{t=1}^T (h_j^t)^2} \sum_{t=1}^T \left( \mathbf{x}^t - \left( \sum_{i \neq j} \mathbf{D}_{*,i} h_i^t \right) \right) h_j^t \end{aligned} \quad (10.22)$$

令：

$$A = \sum_{t=1}^T \mathbf{h}^t \times (\mathbf{h}^t)^T \quad (10.23)$$

$$B = \sum_{t=1}^T \mathbf{x}^t \times (\mathbf{h}^t)^T \quad (10.24)$$

把式(10.23)和式(10.24)代入式(10.22)，可化简得：

$$\mathbf{D}_{*,j} = \frac{1}{A_{j,j}} (B_{*,j} - \mathbf{D}A_{*,j} + \mathbf{D}_{*,j}A_{j,j}) \quad (10.25)$$

最后用式(10.21)对字典进行规则化，这样得到了另一种求解字典最优解的迭代方法。

前面我们固定 $\mathbf{D}$ ，求解出 $\mathbf{h}^t$ 的最优解，如式(10.16)所示。然后同样固定 $\mathbf{h}^t$ ，得到参数 $\mathbf{D}$ 的最优解，如式(10.25)所示。假设的前提是 $\mathbf{h}^t$ 与 $\mathbf{D}$ 是独立的，但事实上这个独立性并不存在，要求解式(10.7)的最优化问题，可以采用EM算法的思想。

- $E$ 步：利用LSTA算法求解出在当前的字典 $\mathbf{D}$ 条件下，最优的隐藏层表示 $\mathbf{h}^t$ 。
- $M$ 步：利用block coordinate descent算法求解出最优的字典 $\mathbf{D}$ 。

重复上面的过程，将得到式(10.7)关于 $\mathbf{h}^t$ 和 $\mathbf{D}$ 的最优解。文献[6]比较详细地分析了稀疏编码的求解过程，读者可自行查阅。此外，前面的讲解都是基于batch的批量最优化求解，对于大数据量的运算效率比较慢，对于稀疏编码的参数实时更新策略，读者可以参考文献[7]。

## 10.5 应用：cifar10图像分类

本节将把前面介绍的三种网络模型：单层感知机、多层感知机和栈式自编码器应用到cifar10图像分类中，cifar10图像集由多伦多大学的Geoffrey Hinton教授等人收集整理。包含了60 000张大小为 $32 \times 32$ 的彩色图像，其中50 000张作为训练数据，10 000张用于测试数据。50 000张训练图片又分为5个子集，分别命名为data\_batch\_1~5。每个子集都包含正好10 000张图片，cifar10包含的图片类别如图10.13所示：

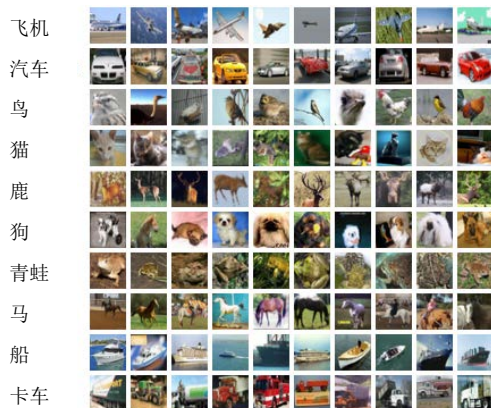


图10.13 cifar10图像集示例

每一个batch图像集被序列化为Python的pickled对象，使用下面的代码来反序列化数据，函数的返回值是一个字典，字典包含了下面的两个元素：

- **data**: 一个大小为 $10\,000 \times 3\,072$ 的numpy数组，每一行代表一张彩色图片数据集，其中前1 024个值代表R通道的数据，后面1 024个值代表G通道数据，最后的1 024个值代表B通道数据。
- **label**: 一个大小为10 000的列表，列表中的每一个元素是0~9范围内的整数值，代表该图像的类别。

```
def unpickle(file):
    import cPickle
    fo = open(file, 'rb')
    dict = cPickle.load(fo)
    fo.close()
    return dict
```

cifar10图像集的baseline为0.1（全部测试数据估值为猫类别），我们分别测试了三种不同的网络模型的分类效果，各模型的超参数设置如图10.14所示：

模 型	超 参 数
感知机	Learning rate: 0.00000001; L1: 0.0, L2: 0.0 epoch: 1000, batch_size: 100
多层感知机	Learning rate: 0.0001; L1: 0.0, L2: 0.01 Hidden Layer size: 共两层，大小分别是1 000和1 000 epoch: 1000, batch_size: 100
栈式自编码器	Learning rate: 0.0001; L1: 0.0, L2: 0.01 Hidden Layer size: 共两层，大小分别是1 000和1 000 batch_size: 100 Pre-training epoch: 200; fine-tune epoch: 1 000

图10.14 三个网络模型的超参数设置

为了体现SAE相比MLP的优势，将栈式自编码器的超参数设置为与多层感知机相同，包括学习率、正则化参数以及隐藏层的神经元设置，然后在此基础上，将SAE的预训练迭代次数设置为200。全部模型运行在GTX 1070显卡上，得到的分类错误率如图10.15所示，读者可以在本书配套的Github网站上下载不同模型的实现代码。

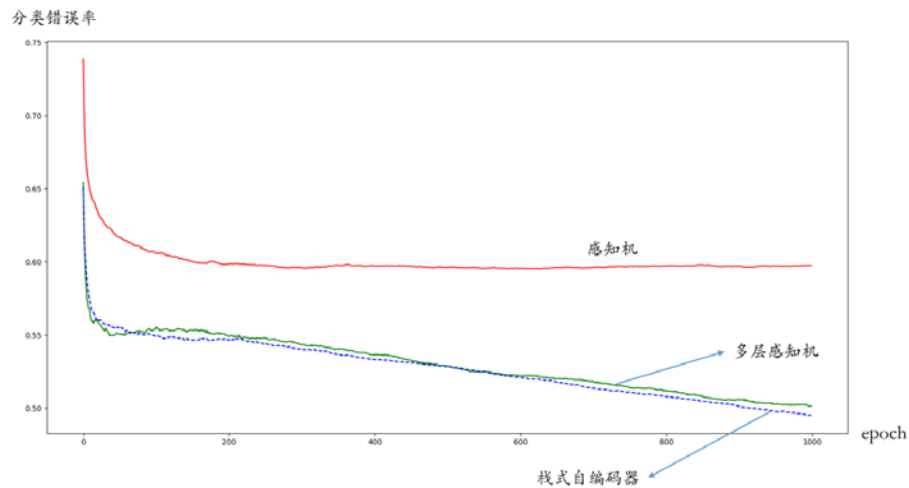


图10.15 按照图10.14进行网络参数设置后，不同模型对应的分类错误率。由此发现，MLP和SAE在性能上都要比单层感知机有了明显进步，而SAE经过预训练的处理后，其效果要略优于MLP

## 参考文献:

---

- [1] Bengio, X. Glorot, Understanding the difficulty of training deep feedforward neural networks, AISTATS 2010.
- [2] G.E. Hinton, S. Osindero, and Y. The. "A fast learning algorithm for deep belief nets", Neural Computation, vol 18, 2006.
- [3] Y.Bengio, Learning deep architectures for AI, Foundations and Trends in Machine Learning 1(2) pages 1–127.
- [4] P Vincent, H Larochelle, Y Bengio. Extracting and composing robust features with denoising autoencoders. Proceedings of the 25th International Conference on Machine Learning. 2008.
- [5] Olshausen, Bruno A; Field, David J (1996). "Emergence of simple-cell receptive field properties by learning a sparse code for natural images" (PDF). Nature. 381 (6583): 607–609.
- [6] Karol Gregor, Yann LeCun. Learning Fast Approximations of Sparse Coding. Proceeding International Conference on Machine Learning. 2010.
- [7] Julien Mairal, Francis Bach, Jean Ponce. Online dictionary learning for sparse coding. International Conference on Machine Learning. 2009.

## 玻尔兹曼机及其相关模型

深度学习研究的对象是具有深层表示的学习模型，它实际上是由两种不同的类型构成：第一种类型是以神经网络为核心的深度神经网络；第二种类型是以概率图模型为核心的深度图模型。前面已经介绍了几种常见的深度神经网络模型，本章将介绍深度学习的另一种形态，即深度概率图模型。

回顾第5章，我们介绍了概率图模型的相关知识点，本章将介绍一种常用的深度图模型——玻尔兹曼机（Boltzmann Machine，简称 BM）。它是马尔科夫无向图网络中的一种，以顶点表示随机变量，边表示变量间的依赖关系。在深度学习领域中，玻尔兹曼机与自编码器一样，是深度学习中常用的预训练模型和无监督学习模型。本章将在第5章的基础上，深入分析玻尔兹曼机，尤其是受限玻尔兹曼机的表示、推断和学习理论，并介绍受限玻尔兹曼机在协同过滤推荐场景下的应用。

在本章中，受限玻尔兹曼机（包括玻尔兹曼机），既可以被看成是神经网络结构，也可以被看成是概率图模型，为了在后面的讨论过程中，让读者不会产生概念上的误解，本章把“神经元”“随机变量”和“节点”这三个概念看成是等价的术语。

### 11.1 玻尔兹曼机

玻尔兹曼机，是一种典型的无向概率图模型，由Geoffrey Hinton和Terry Sejnowski在1985年提出<sup>[2]</sup>。BM网络的结构非常简单，如图11.1所示，BM是一个完全图结构，

所有的结点通过无向边相连，它将节点集划分为可视层节点集 $V$ 和隐藏层节点集 $H$ 。其中可视层，也称为输入层，用于接收可观察的样本数据集合；隐藏层则是对输入数据的抽象，通常能起到隐特征提取、降维等作用。

首先对网络的结构与符号进行定义：对可视层节点集，一般用大写字母 $V$ 来表示， $V$ 的一个具体取值用其小写字母 $v$ 来表示，其中 $v = \{v_1, \dots, v_n\}$ ， $n$ 表示可视层的神经元个数， $v_i$ 表示可视层的第 $i$ 个神经元取值，在图中用空心圆来表示。隐藏层节点集一般用大写字母 $H$ 来表示， $H$ 的一个具体取值用其小写字母 $h$ 来表示，其中 $h = \{h_1, \dots, h_m\}$ ， $m$ 表示隐藏层的神经元个数， $h_j$ 表示隐藏层的第 $j$ 个神经元取值，在图中用实心圆来表示。图11.1展示了一个简单的BM网络结构图，图中由4个可视层神经元和3个隐藏层神经元构成，并且任意两个节点都有边相连，构成一个完全图结构。

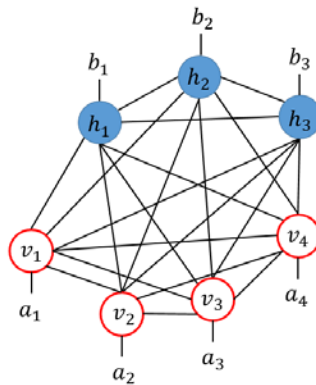


图11.1 玻尔兹曼机

BM具有强大的无监督学习能力，能够学习到输入数据中复杂的规则。但是无差别的全连接结构，使得其训练的代价非常昂贵。因此，在实际的应用中，更多采用的是它的一种简化版变种网络：受限玻尔兹曼机。

为了解决BM网络训练代价高的问题，Paul Smolensky在1986年提出了一种简化的BM结构。在这种网络结构中，仅保留了可视层神经元与隐藏层神经元之间的连接，但可视层神经元之间，以及隐藏层神经元之间互不相连，这样就把网络结构从完全图简化为完全二分图，因为这种网络结构是在玻尔兹曼机的基础上进行一些限制性的改造，因此，把这种网络结构称为受限玻尔兹曼机，或限制玻尔兹曼机（Restricted Boltzmann Machine，简称RBM），图11.2展示了对图11.1进行改造后所对应的RBM网络结构模型。

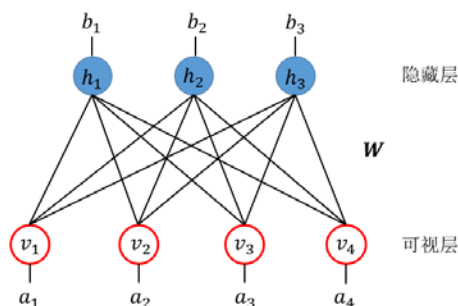


图11.2 受限玻尔兹曼机

RBM的每一个可视层神经元的输入数据类型，可以取二进制数值0或1，也可以取任意实数值。隐藏层单元是用来提取可视层数据的隐式特征，一般是二进制数值，当神经元的值为1的时候，称神经元处于激活状态；当神经元的值为0的时候，称神经元处于非激活或抑制状态。隐藏层单元的取值服从伯努利分布。在本书中，只探讨可视层和隐藏层都为二进制的二值神经网络模型。

上面给出了RBM的网络结构表示，但要确定一个完整的RBM模型，还需要下面的3组参数。

- 可视层神经元与隐藏层神经元的连接**权重参数** $W$ ，如果可视层含有 $n$ 个神经元，隐藏层含有 $m$ 个神经元，则 $W$ 是一个 $n \times m$ 维的矩阵， $w_{ij}$ 表示 $v_i$ 与 $h_j$ 的连接权重。通常也记 $W_{i*} = (w_{i1}, w_{i2}, \dots, w_{im})^T$ 表示所有隐藏层神经元与 $v_i$ 相连的权重参数向量。

$$W = \begin{bmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nm} \end{bmatrix} \quad (11.1)$$

- 可视层神经元的**偏移量参数** $a$ ， $a$ 是一个 $n$ 维向量，其中 $a_i$ 表示可视层神经元 $v_i$ 的偏移量。

$$a = (a_1, a_2, \dots, a_n)^T \quad (11.2)$$

- 隐藏层神经元的**偏移量参数** $b$ ， $b$ 是一个 $m$ 维向量，其中 $b_j$ 表示隐藏层神经元 $h_j$ 的偏移量。

$$b = (b_1, b_2, \dots, b_m)^T \quad (11.3)$$



## 11.2 能量模型

RBM是一个特殊的无向概率图模型，PGM的无向图表示是通过最大团的因子分解来定义，本节也将首先探讨如何构造RBM的最大团的势函数，如果读者对PGM的相关知识点还不太熟悉，请仔细阅读第5章的知识或相关的概率图模型教材<sup>[1]</sup>，本节的很多内容和概念都与PGM紧密相连。

RBM的概率分布通过能量来定义。能量模型（Energy-Based Model）是概率图中一种具有普适意义的模型，能量的概念来源于物理学和热力学，用来描述一个物体的稳定性。我们把能量的概念引入到机器学习领域，并通过下面三个步骤转化为变量的概率分布：首先是定义变量集的能量函数，然后根据能量函数得到每一个子团的势函数，最后根据无向图的马尔科夫独立性把势函数转化为随机变量的概率分布。下面就按照这个步骤来分析从能量函数到概率分布的过程。

### 11.2.1 能量函数

能量模型通过为随机变量集合定义一个能量值（标量），从而捕获该集合中随机变量间的依赖关系。形式化来说，对形如图11.2的RBM网络模型，需要为随机变量集合 $\{v_i, h_j\}$ 分配一个能量值，其表达式如下所示。

$$E(v_i, h_j) = f(w_{ij}, a_i, b_j) \quad (11.4)$$

其中 $\mathbf{W}$ 、 $\mathbf{a}$ 、 $\mathbf{b}$ 的定义请参考式(11.1)到式(11.3)。对于RBM模型，它的能量函数是一个线性模型，式(11.4)的具体形式可以表示为：

$$E(v_i, h_j) = -a_i \times v_i - b_j \times h_j - v_i \times w_{ij} \times h_j \quad (11.5)$$

同理，也可以为全体变量集 $\{v, h\}$ 配置一个能量函数 $f(\mathbf{W}, \mathbf{a}, \mathbf{b})$ ，即：

$$E(v, h) = f(\mathbf{W}, \mathbf{a}, \mathbf{b}) \quad (11.6)$$

仿照式(11.5)的线性表示，把上式转化为：

$$E(v, h) = f(\mathbf{W}, \mathbf{a}, \mathbf{b}) = -\sum_i a_i \times v_i - \sum_j b_j \times h_j - \sum_i \sum_j v_i \times w_{ij} \times h_j \quad (11.7)$$

式(11.7)也可以写成更简洁的矩阵形式：

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{a}^T \times \mathbf{v} - \mathbf{b}^T \times \mathbf{h} - \mathbf{v}^T \times \mathbf{W} \times \mathbf{h} \quad (11.8)$$

现在考察能量函数的意义，从物理学的角度出发，能量越大，则物体越不稳定；反之，能量越小，则物体越稳定。从概率统计的角度来看，如果一个随机变量组合被认为是合理的，则该变量组合也就越稳定，那么它应该具有较少的能量，也就是在所偏好的变量取值上有较小的能量值。

上面的表述从字面来理解有点抽象，下面以式(11.5)为例来说明能量函数的具体意义。

- 固定 $a_i \times v_i$ 和 $b_j \times h_j$ ，考察 $v_i \times w_{ij} \times h_j$ 项。当 $w_{ij} > 0$ 时， $E(v_i = 1, h_j = 1)$ 取最小值，因此此时的能量配置使得网络偏好于 $v_i$ 和 $h_j$ 都取值为1，即 $v_i$ 和 $h_j$ 同时取1的概率比较高。同理，当 $w_{ij} < 0$ 时， $E(v_i = 0, h_j = 0)$ 取最小值，也就是说此时的能量配置使得网络更偏好于 $v_i$ 和 $h_j$ ，至少有一个随机变量取值为0，即 $v_i$ 和 $h_j$ 至少有一个为0的概率比较高。
- 固定 $v_i \times w_{ij} \times h_j$ 和 $b_j \times h_j$ ，考察 $a_i \times v_i$ ， $a_i$ 是对应于可视层神经元 $v_i$ 的偏移量，当 $a_i > 0$ 时，有 $E(v_i = 0, h_j) > E(v_i = 1, h_j)$ ，也就是说此时的能量配置使得网络更偏好于 $v_i$ 的取值为1。当 $a_i < 0$ 时，有 $E(v_i = 1, h_j) > E(v_i = 0, h_j)$ ，此时的能量配置使得网络偏好于 $v_i$ 的取值为0。
- 固定 $v_i \times w_{ij} \times h_j$ 和 $a_i \times v_i$ ，考察 $b_j \times h_j$ ， $b_j$ 是对应于隐藏层神经元 $h_j$ 的偏移量。当 $b_j > 0$ 时，有 $E(v_i, h_j = 0) > E(v_i, h_j = 1)$ ，因此此时的能量配置使得网络偏好于 $h_j$ 的取值为1。当 $b_j < 0$ 时，有 $E(v_i, h_j = 1) > E(v_i, h_j = 0)$ ，因此此时的能量配置使得网络偏好于 $h_j$ 的取值为0。

上面给出了参数的取值对网络偏好的影响，前面的分析是单独考察了式(11.5)的3个不同部分参数对能量值的影响，以及不同取值下的网络偏好，但需要提醒读者的是，这三者间是相互影响的一个整体，上面的例子只是帮助大家更好地理解能量函数的意义，即：**期望在所偏好的变量取值组合上有较小的能量值。**

## 11.2.2 从能量函数到势函数

回忆PGM介绍的势函数概念，无向概率图模型为图中的每一个最大团集合定义了势函数，**势函数是一个非负函数，描述了变量集合间的相互关系**。读者可能已经观察到，势函数概念的表述与能量函数的定义非常相似，唯一的区别在于，能量函数的取值范围可以是任意实数值，而势函数是一个非负函数，为了满足非负性的特点，我们使用指数函数来定义势函数，即：

$$\psi_Q(X_Q) = e^{-E(X_Q)} \quad (11.9)$$

其中 $Q$ 表示图中的一个最大团集合,  $X_Q$ 表示该团集中包含的所有随机变量(神经元)。考察RBM网络的团结构, 不难发现任意一个可视层神经元 $v_i$ 和任意一个隐藏神经元 $h_j$ 构成的集合 $\{v_i, h_j\}$ 就是RBM的最大团, 如图11.3所示。

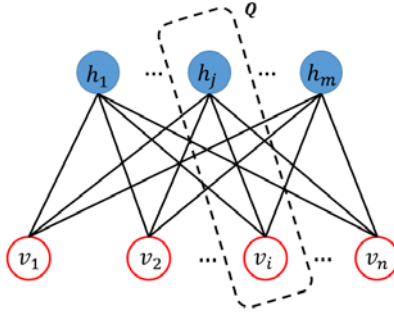


图11.3 RBM的团划分

考察任意一个最大团 $\{v_i, h_j\}, i = 1, 2, \dots, n, j = 1, 2, \dots, m$ , 它的势函数公式可以通过把式(11.5)的能量函数代入式(11.9)得到, 满足:

$$\psi_Q(v_i, h_j) = e^{a_i \times v_i + b_j \times h_j + v_i \times w_{ij} \times h_j} \quad (11.10)$$

### 11.2.3 从势函数到概率分布

在马尔科夫网络中, 多个变量之间的联合概率分布是基于“最大团”分解为多个势函数的乘积。形式化来说, 假设在RBM网络中, 所有团构成的集合为 $\mathcal{C}$ , 对于 $\mathcal{C}$ 中的任意一个最大团 $Q, Q \in \mathcal{C}$ , 它所包含的节点集合记为 $X_Q$ , 模型的联合概率分布 $P(v, h)$ 定义为:

$$P(v, h) = \frac{1}{Z} \prod_{Q \in \mathcal{C}} \psi_Q(X_Q) \quad (11.11)$$

其中 $Z$ 是一个归一化因子, 由于RBM是一个二值神经网络,  $Z$ 的数学形式可以写成:

$$Z = \sum_{v \in \{0,1\}^n} \sum_{h \in \{0,1\}^m} \left( \prod_{Q=(v,h) \in \mathcal{C}} \psi_Q(X_Q) \right) \quad (11.12)$$

$Z$ 的作用在于保证 $P(v, h)$ 的取值满足 $0 \leq P(v, h) \leq 1$ 。由图11.3可知, RBM的所有最大团由任意一个可视层神经元和任意一个隐藏层神经元的二元组 $(v_i, h_j)$ 构成, 因此式

(11.11)的分子可以变为：

$$\prod_{Q \in \mathcal{C}} \psi_Q(X_Q) = \prod_{Q=(v_i, h_j) \in \mathcal{C}} e^{a_i \times v_i + b_j \times h_j + v_i \times w_{ij} \times h_j} = \prod_{i=1}^n \prod_{j=1}^m e^{a_i \times v_i + b_j \times h_j + v_i \times w_{ij} \times h_j} \quad (11.13)$$

其中 $n$ 表示可视层神经元的总数， $m$ 表示隐藏层神经元的总数，由指数函数的幂运算性质，上式可以继续化简为：

$$\prod_{Q \in \mathcal{C}} \psi_Q(X_Q) = \left( \prod_{i=1}^n e^{a_i \times v_i} \right) \times \left( \prod_{j=1}^m e^{b_j \times h_j} \right) \times \left( \prod_{i=1}^n \prod_{j=1}^m e^{v_i \times w_{ij} \times h_j} \right) \quad (11.14)$$

把式(11.14)用矩阵的形式来表示，得到：

$$\prod_{Q \in \mathcal{C}} \psi_Q(X_Q) = e^{a^T \times v} \times e^{b^T \times h} \times e^{v^T \times W \times h} = e^{-E(v, h)} \quad (11.15)$$

把式(11.15)代入式(11.11)可得：

$$P(v, h) = \frac{1}{Z} \prod_{Q \in \mathcal{C}} \psi_Q(X_Q) = \frac{e^{-E(v, h)}}{Z} \quad (11.16)$$

式(11.16)就是RBM的联合概率分布因子分解表示，也被称为**玻尔兹曼分布**。

回顾11.2.1节中在解释能量函数的意义时提到，能量函数使得在所偏好的变量取值上有较小的能量值，而式(11.16)则表明，变量值的概率分布是关于能量函数的减函数，也就是说能量值越小，概率分布越大。从认知的角度来理解，就是说能量越小，则概率分布越大，也就是分布越合理。

### 11.3 推断

前面探讨了RBM的模型表示，以及推导出RBM的联合概率分布，推断的目标就是基于联合概率分布，对随机变量的边缘分布（包括 $p(h)$ 和 $p(v)$ ）和条件概率分布（包括 $p(h|v)$ 和 $p(v|h)$ ）进行查询。

**边缘分布：**对于两层的RBM模型，边缘分布 $p(v)$ 是指只考察可视层神经元集合 $v$ ，并对无关的隐藏层神经元集合 $h = (h_1, h_2, \dots, h_m)$ 进行求和后得到的结果。同理，边缘分布 $p(h)$ 是指只考察隐藏层神经元集合 $h$ ，对无关的可视层神经元集合 $v = (v_1, v_2, \dots, v_n)$ 进行求和后得到的结果。在实际的应用中，边缘推断可以帮助我们

对输入数据进行重构：即假设输入数据的真实分布服从 $p(v)$ ，这是一个未知值，通过RBM的训练，构造出输入数据的近似概率分布为 $q(v)$ ，训练目标是使得 $p(v)$ 与 $q(v)$ 尽量的接近，当训练收敛时，RBM可以作为一个生成模型，利用学习得到的预测概率分布 $q(v)$ 来生成新的样本数据集。

**条件分布：**条件分布 $p(h|v)$ 是指当给定某一输入数据或观察序列 $v$ 的条件下，隐藏层神经元的取值为 $h$ 的概率。同理，条件分布 $p(v|h)$ 是指当给定某一隐藏序列 $h$ 的条件下，可视层神经元的取值为 $v$ 的概率。条件推断的一个很重要的应用就是寻找隐因子或者对输入的观察序列进行降维，通常来说，隐藏层的神经元都比可视层神经元要少，利用条件推断，可以得到输入数据 $v$ 的隐藏层表示 $h$ 。

下面来详细探讨边缘分布和条件分布的相关推导过程。

### 11.3.1 边缘分布

在无监督学习中，接收到的输入数据通常是没有任何标签的，给出任意的训练数据集集合，应该如何合理地预测数据的原始生成概率分布呢？这就是本节要讨论的边缘分布问题。由边缘分布概率公式，有：

$$p(v) = \sum_h p(v, h) = \sum_{h \in \{0,1\}^m} \frac{e^{-E(v,h)}}{Z} \quad (11.17)$$

其中 $Z$ 是归一化因子，它的值为 $Z = \sum_v \sum_h e^{-E(v,h)}$ ，首先来化简(11.17)式的分子：

$$\begin{aligned} \sum_{h \in \{0,1\}^m} e^{-E(v,h)} &= \sum_{h \in \{0,1\}^m} e^{a^T \times v + b^T \times h + v^T \times W \times h} \\ &= e^{a^T \times v} \times \sum_{h_1 \in \{0,1\}} \dots \sum_{h_m \in \{0,1\}} e^{b^T \times h + v^T \times W \times h} \\ &= e^{a^T \times v} \times \sum_{h_1 \in \{0,1\}} \dots \sum_{h_m \in \{0,1\}} e^{\sum_j b_j \times h_j + v^T \times W_{*,j} \times h_j} \\ &= e^{a^T \times v} \times \sum_{h_1 \in \{0,1\}} \dots \sum_{h_m \in \{0,1\}} \left( \prod_{j=1}^m e^{b_j \times h_j + v^T \times w_{*,j} \times h_j} \right) \\ &= e^{a^T \times v} \times \left( \sum_{h_1 \in \{0,1\}} e^{b_1 \times h_1 + v^T \times w_{*,1} \times h_1} \right) \dots \end{aligned}$$

$$\left( \sum_{h_m \in \{0,1\}} e^{b_m \times h_m + v^T \times w_{*,m} \times h_m} \right) \quad (11.18)$$

式(11.18)中每一个 $h_j$ 的取值只能取0或1，故：

$$\sum_{h_j \in \{0,1\}} e^{b_j \times h_j + v^T \times w_{*,j} \times h_j} = 1 + e^{b_j + v^T \times w_{*,j}} \quad (11.19)$$

把式(11.19)代入式(11.18)，可得：

$$\begin{aligned} \sum_{h \in \{0,1\}^m} e^{-E(v,h)} &= e^{a^T \times v} \\ &\times \left( \sum_{h_1 \in \{0,1\}} e^{b_1 \times h_1 + v^T \times w_{*,1} \times h_1} \right) \dots \dots \left( \sum_{h_m \in \{0,1\}} e^{b_m \times h_m + v^T \times w_{*,m} \times h_m} \right) \\ &= e^{a^T \times v} \times (1 + e^{b_1 + v^T \times w_{*,1}}) \dots \dots (1 + e^{b_m + v^T \times w_{*,m}}) \\ &= e^{a^T \times v} \times \prod_{j=1}^m (1 + e^{b_j + v^T \times w_{*,j}}) = e^{a^T \times v} \times e^{\sum_{j=1}^m \ln(1 + e^{b_j + v^T \times w_{*,j}})} \\ &= e^{a^T \times v + \sum_{j=1}^m \ln(1 + e^{b_j + v^T \times w_{*,j}})} \end{aligned} \quad (11.20)$$

把式(11.20)代入式(11.17)，即可得到边缘分布：

$$p(v) = \frac{e^{a^T \times v + \sum_{j=1}^m \ln(1 + e^{b_j + v^T \times w_{*,j}})}}{Z} \quad (11.21)$$

比较式(11.21)与式(11.16)，可以发现两者的表达形式很相似，为了统一，定义**自由能量** $F(v)$ ，满足：

$$F(v) = -a^T \times v - \sum_{j=1}^m \ln(1 + e^{b_j + v^T \times w_{*,j}}) \quad (11.22)$$

因此，边缘分布 $p(v)$ 也可以简写为：

$$p(v) = \sum_{h \in \{0,1\}^m} \frac{e^{-E(v,h)}}{Z} = \frac{e^{-F(v)}}{Z} \quad (11.23)$$

式(11.23)就是可视层节点集的边缘概率分布，读者也可以按照上面的推导过程来求解隐藏层节点集的边缘概率分布 $p(h)$ ，这里我们不再加以详述。(11.21)式分子中带有表达式项 $\ln(1 + e^{b_j + v^T \times w_{*,j}})$ ，该函数恰好是一个softplus函数，也就是满足：

$$\text{softplus}(b_j + v^T \times w_{*,j}) = \ln(1 + e^{b_j + v^T \times w_{*,j}})$$

在第8章中，曾经提到它是神经网络中ReLU激活函数的一种平滑近似，其函数图像如图11.4所示。

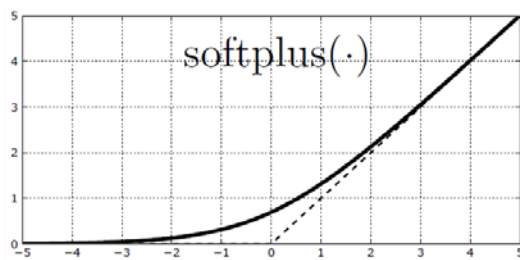


图11.4 softplus函数

### 11.3.2 条件分布

首先来考察当固定可视层节点数据  $v = (v_1, v_2, \dots, v_n)$  的条件下，隐藏层节点的取值为  $h = (h_1, h_2, \dots, h_m)$  的条件概率  $p(h|v)$ ，在后面的章节中，我们也把由  $v$  推导出  $h$  的过程称为**编码（encoding）**。

RBM网络是一个无向概率图模型，因此它也满足成对马尔科夫独立性，也就是说：给定两个随机变量子集的D-分离集，在给定分离集数据的条件下，这两个随机变量子集条件独立。考察图11.5的RBM网络。

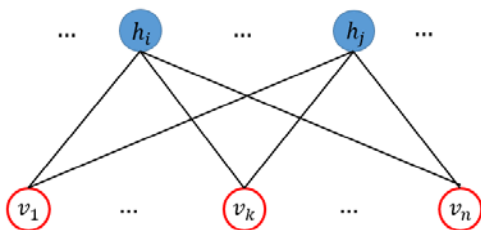


图11.5 RBM网络

对于任意的隐藏层神经元  $h_i$  和  $h_j$ ，它们的分离集就是所有的可视层神经元集合，即  $V = \{v_1, \dots, v_n\}$ ，对隐藏层的任意两个神经元  $h_i$  和  $h_j$ ，从  $h_i$  到  $h_j$  的最短路径都必然经过该分离集节点，如图11.6所示。

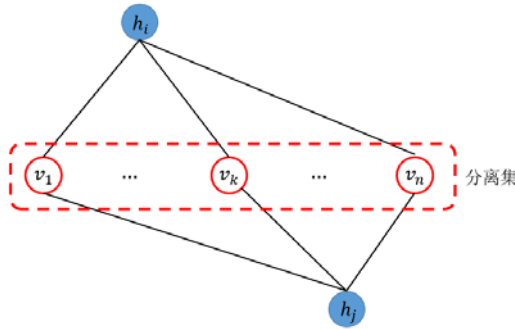


图11.6 通过可视层节点集把隐藏层节点分离，任意两个隐藏层神经元的最短距离均为2，

最短路径的表达形式为： $h_i \rightarrow v_k \rightarrow h_j$

由5.4.2节介绍的全局马尔科夫独立性，可知条件分布满足：

$$p(h|v) = \prod_{j=1}^m p(h_j|v) \quad (11.24)$$

下面推导 $p(h_j|v)$ 的具体表示形式，根据式(11.16)的联合分布公式，可以将条件分布 $p(h|v)$ 化简为：

$$p(h|v) = \frac{p(v, h)}{\sum_{h^*} p(v, h^*)} = \frac{e^{a^T \times v + b^T \times h + v^T \times W \times h} / Z}{\sum_{h^*} (e^{a^T \times v + b^T \times h^* + v^T \times W \times h^*} / Z)} \quad (11.25)$$

其中  $h^*$  是隐藏层节点集的任意可能取值， $h^* \in \{0,1\}^m$ ，且  $h^* = (h_1^*, h_2^*, \dots, h_m^*)$ ， $h_i^* \in \{0,1\}$ ，利用指数函数的性质，可以进一步化简：

$$\begin{aligned} p(h|v) &= \frac{e^{a^T \times v + b^T \times h + v^T \times W \times h} / Z}{\sum_{h^*} (e^{a^T \times v + b^T \times h^* + v^T \times W \times h^*} / Z)} \\ &= \frac{e^{\sum_j b_j \times h_j + v^T \times W_{*,j} \times h_j}}{\sum_{h_1^* \in \{0,1\}} \sum_{h_2^* \in \{0,1\}} \dots \sum_{h_m^* \in \{0,1\}} (e^{\sum_j b_j \times h_j^* + v^T \times W_{*,j} \times h_j^*})} \\ &= \frac{\prod_j e^{b_j \times h_j + v^T \times W_{*,j} \times h_j}}{\sum_{h_1^* \in \{0,1\}} \sum_{h_2^* \in \{0,1\}} \dots \sum_{h_m^* \in \{0,1\}} (\prod_j e^{b_j \times h_j^* + v^T \times W_{*,j} \times h_j^*})} \\ &= \frac{\prod_j e^{b_j \times h_j + v^T \times W_{*,j} \times h_j}}{(\sum_{h_1^* \in \{0,1\}} (e^{b_1 \times h_1^* + v^T \times W_{*,1} \times h_1^*})) \dots (\sum_{h_m^* \in \{0,1\}} (e^{b_m \times h_m^* + v^T \times W_{*,m} \times h_m^*}))} \end{aligned}$$



$$= \frac{\prod_j e^{b_j \times h_j + v^T \times W_{*,j} \times h_j}}{\prod_j \left( \sum_{h_j^* \in \{0,1\}} (e^{b_j \times h_j^* + v^T \times W_{*,j} \times h_j^*}) \right)} \quad (11.26)$$

考察式(11.26)的分母, 由于 $h_j^*$ 的取值只能是1或0, 当 $h_j^* = 0$ 时:

$$e^{b_j \times h_j^* + v^T \times W_{*,j} \times h_j^*} = 1 \quad (11.27)$$

当 $h_j^* = 1$ 时:

$$e^{b_j \times h_j^* + v^T \times W_{*,j} \times h_j^*} = e^{b_j + v^T \times W_{*,j}} \quad (11.28)$$

把式(11.27)和式(11.28)代入式(11.26), 可得:

$$\begin{aligned} p(h|v) &= \frac{\prod_j e^{b_j \times h_j + v^T \times W_{*,j} \times h_j}}{\prod_j \left( \sum_{h_j^* \in \{0,1\}} (e^{b_j \times h_j^* + v^T \times W_{*,j} \times h_j^*}) \right)} \\ &= \frac{\prod_j e^{b_j \times h_j + v^T \times W_{*,j} \times h_j}}{\prod_j (1 + e^{b_j + v^T \times W_{*,j}})} = \prod_j \frac{e^{b_j \times h_j + v^T \times W_{*,j} \times h_j}}{(1 + e^{b_j + v^T \times W_{*,j}})} \end{aligned} \quad (11.29)$$

把式(11.24)和式(11.29)进行比较, 可以得到每一个隐藏层神经元的条件概率:

$$p(h_j|v) = \frac{e^{b_j \times h_j + v^T \times W_{*,j} \times h_j}}{1 + e^{b_j + v^T \times W_{*,j}}} \quad (11.30)$$

当 $h_j = 0$ 时:

$$p(h_j = 0|v) = \frac{1}{1 + e^{b_j + v^T \times W_{*,j}}} \quad (11.31)$$

当 $h_j = 1$ 时:

$$p(h_j = 1|v) = \frac{e^{b_j + v^T \times W_{*,j}}}{1 + e^{b_j + v^T \times W_{*,j}}} = \frac{1}{1 + e^{-b_j - v^T \times W_{*,j}}} \quad (11.32)$$

由式(11.32)可得, 每一个隐藏层神经元取值的概率分布服从伯努利分布, 且 $p(h_j = 1|v)$ 的取值是一个sigmoid函数, 记为 $p(h_j = 1|v) = \text{sigm}(b_j + v^T \times W_{*,j})$ 。

同理, 仿照上面的推导过程, 当给定隐藏层节点数据 $h = (h_1, h_2, \dots, h_m)$ 的条件下, 可视层节点的取值为 $v = (v_1, v_2, \dots, v_n)$ 的条件概率 $p(v|h)$ , 也把由 $h$ 推导出 $v$ 的过程称为**解码或重构 (reconstruct)**。

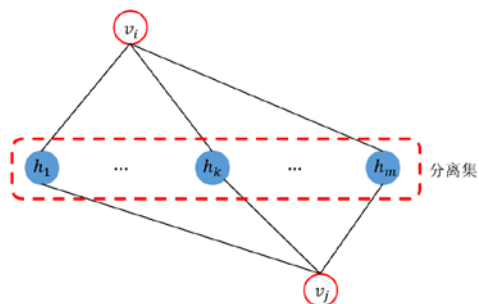


图11.7 参照图11.6, 通过隐藏层节点集把可视层节点分离, 任意两个可视层神经元的最短距离均为 2, 最短路径的表达形式为:  $v_i \rightarrow h_k \rightarrow v_j$

条件分布同样满足全局马尔科夫独立性:

$$p(v|h) = \prod_{i=1}^n p(v_i|h) \quad (11.33)$$

不加推导直接给出可视层节点的条件概率公式, 读者可以按照前面的推导过程自行推导。

每一个可视层神经元的条件概率取值为:

$$p(v_i|h) = \frac{e^{a_i \times v_i + v_i \times W_{i,*} \times h}}{1 + e^{a_i + v_i \times W_{i,*} \times h}} \quad (11.34)$$

当  $v_i = 0$  时:

$$p(v_i = 0|h) = \frac{1}{1 + e^{a_i + v_i \times W_{i,*} \times h}} \quad (11.35)$$

当  $h_j = 1$  时:

$$p(v_i = 1|h) = \frac{1}{1 + e^{-a_i - W_{i,*} \times h}} \quad (11.36)$$

每一个可视层神经元取值的概率分布服从伯努利分布, 且  $p(v_i = 1|h)$  的取值是一个 sigmoid 函数, 记为  $p(v_i = 1|h) = \text{sigm}(a_i + W_{i,*} \times h)$ 。

## 11.4 学习

经过前面的表示理论和推断理论分析, 我们对RBM的结构和用途有了更深入的了解, 那么现在考察如何有效学习网络参数。形式化地说, 当给定一个初始化的RBM网

络模型，以及 $s$ 个训练样本数据，记为 $\{v^1, v^2, \dots, v^s\}$ ，（注：读者应该区分 $v^i$ 与 $v_i$ 的区别，带上标的 $v^i$ 表示第 $i$ 个训练样本数据， $v_i$ 则表示任意一个样本数据 $v$ 中第 $i$ 个神经元的取值），RBM参数学习的目标就是训练合适的参数值 $W$ 、 $a$ 、 $b$ ，使得RBM网络模型能够尽可能地模拟真实的概率分布。

### 11.4.1 最大似然估计

如前所述，RBM是一个能量模型，如果一个随机变量组合被认为是合理的，那么它对应的能量函数应该具有较小的函数值，也就是具有较大的概率分布。因此，当接收到 $s$ 个样本数据 $(v^1, v^2, \dots, v^s)$ ，因为它们都是已经真实存在的数据，即数据是合理的，那么每一个样本 $v^i$ 在RBM中都应该具有较大的发生概率 $p(v^i)$ 。

这样学习目标就可以形式化为最大化所有输入样本的概率。

$$\max_{\theta} \prod_{i=1}^s p(v^i) \quad (11.37)$$

由于 $p(v^i) \leq 1$ ，多个 $p(v^i)$ 直接相乘的结果可能会变得非常小，考虑到浮点数运算的精度问题，因此，在实际应用中，通常对(11.37)式加入对数进行求解。此外，最优优化问题一般都转化为求解最小值，因此目标函数可以转化为：

$$\min_{\theta} -\ln \left( \prod_{i=1}^s p(v^i) \right) = \min_{\theta} -\sum_{i=1}^s \ln p(v^i) \quad (11.38)$$

利用梯度下降算法(Gradient Descent)来求解式(11.38)的最优化问题，回顾第7章，梯度下降的核心问题是求取参数的导数，而11.3.1节已经给出了边缘分布 $p(v)$ 的定义，把式(11.23)代入式(11.38)，可得：

$$\ln p(v) = \ln \frac{e^{-F(v)}}{Z} = -F(v) - \ln Z$$

对上式的两边分别对参数求导，则相应的求导公式可化为：

$$\frac{\partial \ln p(v)}{\partial \theta} = \frac{\partial (-F(v))}{\partial \theta} - \frac{\partial (\ln Z)}{\partial \theta} \quad (11.39)$$

考察式(11.39)，式子由两部分构成，其中第一部分 $\frac{\partial (-F(v))}{\partial \theta}$ 称为正梯度，第二部分 $\frac{\partial (\ln Z)}{\partial \theta}$ 称为负梯度。首先考察正梯度的计算。

$$\frac{\partial(-F(v))}{\partial \theta} = \frac{\partial \left( a^T \times v + \sum_{j=1}^m \ln(1 + e^{b_j + v^T \times w_{*,j}}) \right)}{\partial \theta} \quad (11.40)$$

• 若  $\theta = w_{ij}$ , 式(11.40)可以化简为:

$$\frac{\partial(-F(v))}{\partial w_{ij}} = \frac{\partial \left( a^T \times v + \sum_{j=1}^m \ln(1 + e^{b_j + v^T \times w_{*,j}}) \right)}{\partial w_{ij}} = \frac{e^{b_j + v^T \times w_{*,j}}}{1 + e^{b_j + v^T \times w_{*,j}}} \times v_i \quad (11.41)$$

由式(11.36), 知道  $\frac{e^{b_j + v^T \times w_{*,j}}}{1 + e^{b_j + v^T \times w_{*,j}}}$  恰好等于条件分布  $p(h_j = 1|v)$ , 把其代入式(11.41)可得:

$$\frac{\partial(-F(v))}{\partial w_{ij}} = p(h_j = 1|v) \times v_i \quad (11.42)$$

• 若  $\theta = a_i$ , 式(11.40)可以化简为:

$$\frac{\partial(-F(v))}{\partial a_i} = \frac{\partial \left( a^T \times v + \sum_{j=1}^m \ln(1 + e^{b_j + v^T \times w_{*,j}}) \right)}{\partial a_i} = v_i \quad (11.43)$$

• 若  $\theta = b_j$ , 式(11.40)可以化简为:

$$\frac{\partial(-F(v))}{\partial b_j} = \frac{\partial \left( a^T \times v + \sum_{j=1}^m \ln(1 + e^{b_j + v^T \times w_{*,j}}) \right)}{\partial b_j} = p(h_j = 1|v) \quad (11.44)$$

下面我们来重点关注负梯度的求解过程:

$$\begin{aligned} \frac{\partial \ln Z}{\partial \theta} &= \frac{\partial \ln \sum_v \sum_h e^{-E(v,h)}}{\partial \theta} = \frac{\sum_v \sum_h \left( e^{-E(v,h)} \times \frac{\partial(-E(v,h))}{\partial \theta} \right)}{\sum_v \sum_h e^{-E(v,h)}} \\ &= \sum_v \sum_h \frac{e^{-E(v,h)} \times \frac{\partial(-E(v,h))}{\partial \theta}}{\sum_v \sum_h e^{-E(v,h)}} \\ &= \sum_v \sum_h \left( p(v,h) \times \frac{\partial(-E(v,h))}{\partial \theta} \right) \\ &= \sum_v p(v) \left( \sum_h \left( p(h|v) \times \frac{\partial(-E(v,h))}{\partial \theta} \right) \right) \end{aligned} \quad (11.45)$$

注意观察式(11.45)的括号内层求和公式, 可以发现下面的等式成立:

$$\sum_h \left( p(h|v) \times \frac{\partial(-E(v, h))}{\partial \theta} \right) = \frac{\sum_h \left( e^{-E(v, h)} \times \frac{\partial(-E(v, h))}{\partial \theta} \right)}{\sum_h e^{-E(v, h)}} = \frac{\partial(-F(v))}{\partial \theta} \quad (11.46)$$

其中最后一个等式成立，是由于 $e^{-F(v)} = \sum_h e^{-E(v, h)}$ 成立，读者可自行验证，这里不再详述。把式(11.46)代入式(11.45)可得：

$$\frac{\partial \ln Z}{\partial \theta} = \sum_v p(v) \times \frac{\partial(-F(v))}{\partial \theta} \quad (11.47)$$

- 若 $\theta = w_{ij}$ ，把式(11.42)代入式(11.47)可得：

$$\frac{\partial \ln Z}{\partial w_{ij}} = \sum_v p(v) \times p(h_j = 1|v) \times v_i \quad (11.48)$$

- 若 $\theta = a_i$ ，把式(11.43)代入式(11.47)可得：

$$\frac{\partial \ln Z}{\partial a_i} = \sum_v p(v) \times v_i \quad (11.49)$$

- 若 $\theta = b_j$ ，把式(11.44)代入式(11.47)可得：

$$\frac{\partial \ln Z}{\partial b_j} = \sum_v p(v) \times p(h_j = 1|v) \quad (11.50)$$

至此目标函数式(11.38)的正负梯度求取完毕，把正梯度对应的求导公式(11.42)、公式(11.43)、公式(11.44)和负梯度对应的求导公式(11.48)、公式(11.49)、公式(11.50)相结合，得到最终的参数梯度如下：

$$\frac{\partial \ln p(v)}{\partial w_{ij}} = p(h_j = 1|v) \times v_i - \sum_{v^*} p(v^*) \times p(h_j = 1|v^*) \times v_i^* \quad (11.51)$$

$$\frac{\partial \ln p(v)}{\partial a_i} = v_i - \sum_{v^*} p(v^*) \times v_i^* \quad (11.52)$$

$$\frac{\partial \ln p(v)}{\partial b_j} = p(h_j = 1|v) - \sum_{v^*} p(v^*) \times p(h_j = 1|v^*) \quad (11.53)$$

对于正梯度来说，它的求解仅依赖于样本数据，但负梯度的计算要复杂得多，它需要对所有的 $v$ 值数据进行求和， $v = \{0,1\}^n$ ，也就是一共有 $2^n$ 种可能，只有当 $n$ 的值足够小时，才具有实用价值。

### 11.4.2 对比散度

由于RBM的参数学习中，通过暴力方法求解负梯度具有指数级的时间复杂度，因此，很难将算法应用于实践中。我们考察负梯度的求解过程，由式(11.47)：

$$\frac{\partial \ln Z}{\partial \theta} = \sum_v p(v) \times \frac{\partial(-F(v))}{\partial \theta} = E_v \left( \frac{\partial(-F(v))}{\partial \theta} \right) \quad (11.54)$$

负梯度求导实际上是一个求期望值的过程，一种可行的方案是，如果以分布 $p(v)$ 来采样 $v$ ，得到 $m$ 个不同的采样集 $\{v^1, v^2, \dots, v^m\}$ ，那么可以把这 $m$ 个不同的采样数据近似等于 $v$ 的全部可能取值，这样式(11.51)、式(11.52)和式(11.53)就可以分别改写为：

$$\frac{\partial \ln p(v)}{\partial w_{ij}} = p(h_j = 1|v) \times v_i - \sum_{t=1}^m (p(h_j = 1|v^t) \times v_i^t) \quad (11.55)$$

$$\frac{\partial \ln p(v)}{\partial a_i} = v_i - \sum_{t=1}^m v_i^t \quad (11.56)$$

$$\frac{\partial \ln p(v)}{\partial b_j} = p(h_j = 1|v) - \sum_{t=1}^m p(h_j = 1|v^t) \quad (11.57)$$

这样问题的关键是如何获取这 $m$ 个样本数据。我们采用Gibbs采样来获取，由算法5.5可知，Gibbs采样是高维分布中的近似采样算法，对于给定的初始状态数据 $v^0$ 和 $h^0$ ，分别以下面的公式进行采样：

$$h^{t+1} \sim p(h^{t+1}|v^t) = \text{sigmoid}(Wv^t + b) \quad (11.58)$$

$$v^{t+1} \sim p(v^{t+1}|h^{t+1}) = \text{sigmoid}(Wh^{t+1} + a) \quad (11.59)$$

其采样的可视化过程如图11.8所示，经过有限步的采样后，所得到的数据 $(v^t, h^t)$ 服从分布 $p(v, h)$ 。

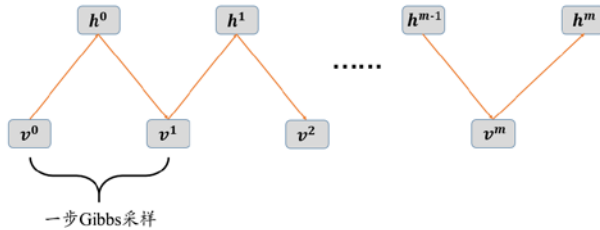


图11.8 Gibbs采样示意图

Gibbs采样虽然相比于暴力求解负梯度的方法，时间复杂度提升很大，但是由于



## 11.5 应用：个性化推荐

今天的互联网产品中都广泛采用了个性化推荐的技术，精准推荐已经成为大数据领域中一项非常前沿和热门的研究工作。一方面，这是由于随着内容的爆炸性增长，用户很少时间和精力对物品进行有效的筛选和甄别，如果能够在海量的数据中帮助用户找到感兴趣的物品，无疑能够极大地提升产品的用户活跃度和用户的粘性。另一方面，在移动互联网时代，用户查找感兴趣物品的方式已经发生了很多的改变，传统的方法，通常包括通过搜索引擎查找，或者通过网站的分类导航目录来查找，但在移动设备中，由于受到界面尺寸大小的限制，上面两种查找方式在交互上都不太友好，因此，给用户主动推送感兴趣的物品，俨然成为了一种可行且高效的方案。

深度学习近年来在理论和工程领域都得到了快速的发展，并且在图像处理、计算机视觉、自然语言处理等领域都取得了不俗的效果，因此把深度学习技术应用于个性化推荐已经成为人工智能新的发展方向。把深度学习应用于个性化推荐领域最早始于2006年，由Salakhutdinov和Hinton等人把RBM模型应用于Netflix的个性化推荐竞赛中<sup>[6]</sup>。本节将介绍个性化推荐相关的知识，包括推荐架构的设计和常用的推荐算法，重点考察如何将RBM模型应用于协同过滤推荐中。

### 11.5.1 个性化推荐概述

互联网技术的快速发展，使得我们早已从过去的信息匮乏的年代进入到信息过载的时代，它体现在：一方面，用户需要在海量的数据中找到自己喜欢的物品。另一方面，物品的生产厂家也希望物品能被精准投放到合适的用户群体中，以实现利益最大化。推荐系统，则是连接用户和物品之间的桥梁，个性化推荐的形式化定义可表述为：在特定的推荐产品形态或推荐场景下，构造合理的算法模型与系统架构，把正确的物品，在正确的时间，推送给正确的用户。个性化推荐的应用场景非常广泛，包括：内容产业、电子商务、社交网络、广告精准投放等领域。

#### 1. 音乐、视频、新闻等内容场景推荐

以音乐、视频和新闻为代表的內容产业，是个性化推荐技术应用相当广泛的领域。这是由于在内容场景下进行个性化推荐，有其天然的必要性。以音乐推荐为例，当前主流的音乐平台，如国内的QQ音乐、酷狗，国外的Spotify、Pandora等，都有上千万首歌曲的曲库规模，但这些歌曲的播放率往往存在非常严重的长尾效应。从歌曲的角



度来说, 1%的歌曲的播放量占全平台播放量的80%以上; 从用户的角度来说, 很多时候, 用户收听的歌曲都局限在一个很小的范围内, 这样就造成了一个矛盾, 一方面是歌曲数量很多, 但利用率不高, 另一方面, 用户经常感觉收听的歌曲没有惊喜, 对新鲜感的要求较高。图11.10展示了Spotify的Discover Weekly推荐场景, Discover Weekly根据用户的历史行为喜好来推送歌曲, 被评为2015年全球最具颠覆性的40大创新之一。

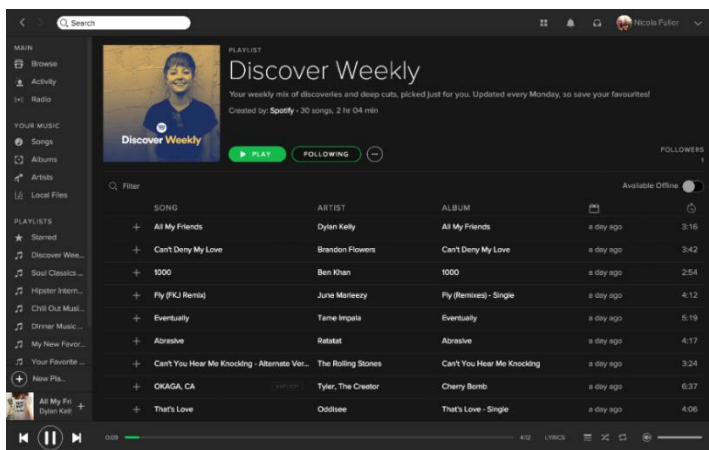


图11.10 Spotify的Discover Weekly被评为2015年全球最具颠覆性的40大创新之一

( 图片截取自Spotify )

## 2. 电子商务

在电子商务中, 推荐系统不但可以作为帮助用户发现喜好物品的工具, 更是能够将这种推荐转化为营收。著名的电子商务网站亚马逊 (Amazon) 就是个性化推荐系统的积极应用者和推广者, 如图11.11所示。据统计, 用户在亚马逊购买的商品, 有超过30%的比例是从个性化推荐转化而来。亚马逊的推荐系统, 使用的主要算法是协同过滤, 也就是根据用户之间的购买行为, 找出具有相似购买群体的物品, 或者具有相似购买行为的用户。



图11.11 Amazon的推荐列表, 根据用户的购买行为进行推荐 ( 图片截取自Amazon )

3. 社交网络

与前面的两个场景不同，社交网络关注的更多是人与人之间的关系，如Facebook、Twitter等。在社交网络中，好友之间存在互动，如分享、评论、转发等操作，使得社区的推荐具有一种双向性，这种双向性与人和物品的交互不同，物品的推荐在绝大部分情况下都是用户单向发起的行为，并由用户的行为数据来判断是否喜欢被推荐的物品，而社交网络中，人与人之间可以相互互动，更能准确反映两者之间的亲密度。

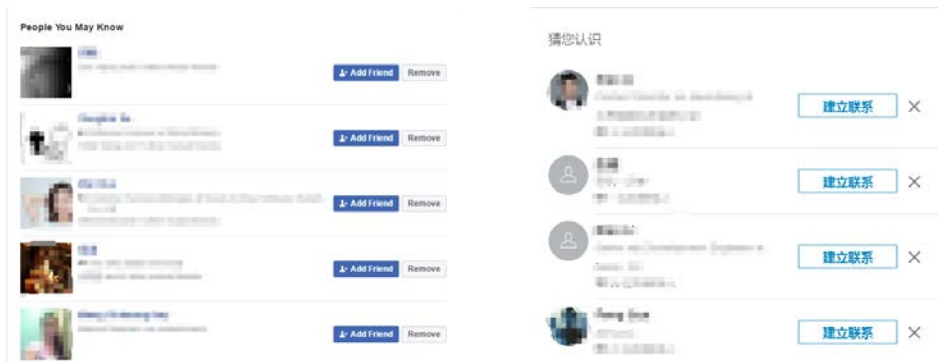


图11.12 Facebook（左）和Linkedin（右）的好友推荐，根据相似用户来进行好友推荐（图片截取自Facebook和Linkedin）

4. 广告精准投放

广告的精准投放是很多互联网公司的收入来源。从理论上来说，个性化广告与个性化推荐本质上是一致的，在个性化广告场景下，物品就是指广告，但它们两者之间有一个比较明显的区别，那就是在个性化推荐中，更关心的是如何帮助用户找到感兴趣的物品，但在个性化广告中，更关心的是广告主的广告是否投放到合适的用户中，并且是否带动了CTR等指标的提升。简单来说，个性化推荐是以人为核心，而个性化广告则更多是以广告为核心。

个性化广告的投放维度非常多，既包括对用户的喜好建模，同时也受到各种条件的约束，如国家、地区、用户的年龄、性别等人口统计学信息。

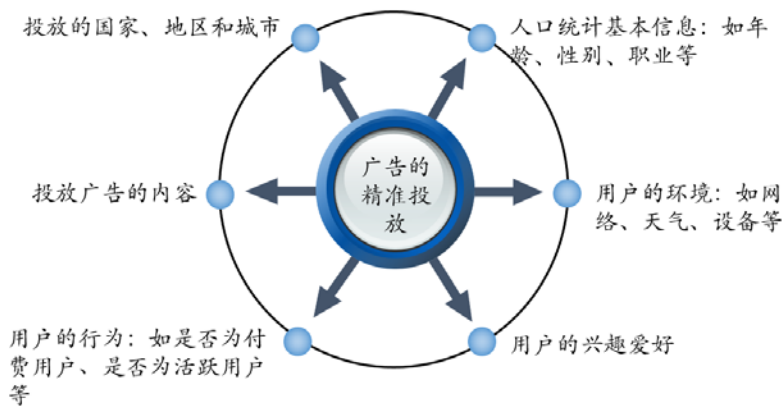


图11.13 广告精准投放架构

11.5.2 个性化推荐架构与算法

一个完备的推荐系统,通常是由数据、产品、架构和算法四大要素组成,如图11.14所示,其中架构和算法又是最重要的两个方面:架构是指数据的处理流程,从用户的角度来看,推荐架构相当于一个黑盒,当用户通过向客户端发送请求后,请求经过黑盒的处理,最终得到推送的数据返回给用户。从个性化推荐的角度,推荐架构控制着数据的流动,协调着不同推荐策略之间的切换,因此,构建合理的推荐架构是推荐系统的基础;另一方面,各种推荐算法模型构成了推荐系统的核心,它们控制着最终结果数据的生成。本节将分别阐述推荐架构的搭建和常用的算法策略。



图11.14 精准推荐的四要素：数据、产品、推荐算法和推荐架构

1. 推荐架构

根据功能和处理方式的不同,个性化推荐架构通常由离线层、计算层和实时层三

大部分构成。

**离线层 (Offline):** 主要负责基础数据的计算任务，这类任务的特点是数据量和计算量都特别巨大，但是不需要实时响应，因此比较适合按周期进行离线更新。通常由于数据量过大，往往需要依托于大规模的计算机集群，如Hadoop、Spark等分布式平台。离线层常见的处理任务包括数据流水的预处理，大规模机器学习模型的训练等。

**实时层 (Online):** 主要用于接收用户的实时行为并对行为做出响应，这类任务的特点与离线层刚好相反，处理的数据少但需要实时反馈。比如在电商推荐中，用户的实时购买行为、实时收藏行为等敏感操作都含有非常丰富的信息。由用户兴趣的半衰期模型可知，如式(11.60)所示，随着时间的变化，用户的兴趣会逐渐发生转移，距离当前时间越近的操作更能反映用户当前的心理行为。因此，实时层对于捕获并响应这一类操作起到至关重要的作用，可以与离线层形成数据互补。

$$f(t) = A + \frac{B}{\exp(kt)} \tag{11.60}$$

**计算层:** 有些文献也称为近线层 (Nearline)，计算层的任务是接收客户端的推荐请求，并依据实时层和离线层的基础数据，选择合适的推荐策略，将推送数据返回给客户端。计算层的推荐过程就像一个最优化过程，离线层产生所有可推送的候选数据源，实时层相当于为推送策略添加了约束条件，如何在满足用户当前实时操作的条件下，从所有可用的候选数据空间中挑选最合适的数据返回给用户。

图11.15展示了一个典型的个性化推荐的系统架构设计。

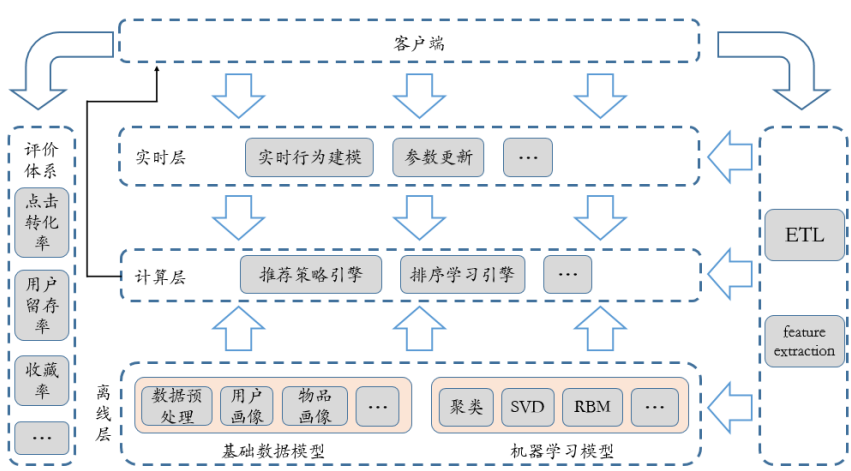


图11.15 个性化系统的架构设计

## 2. 算法

推荐算法是个性化推荐的核心，经过近20年的发展，个性化推荐算法在理论领域和工程领域都取得了非常重大的进步。下面介绍常用的两种推荐模型，即基于内容的推荐和基于协同过滤的推荐。这里需要注意的是，这是两类算法模型的总称，并不是具体的单一算法，例如，基于协同过滤的推荐，它包括了很多种不同的算法实现，如矩阵分解（SVD）、神经网络（RBM、RNN）和图算法等。

**基于内容的推荐（Content-Based Recommendations，简称CB算法）：**是应用最早的推荐算法之一。CB算法通过特征工程技术，为用户和物品分别构建用户画像（user profile）和物品画像，根据用户的特征数据，为用户推荐与其特征相似的物品。CB算法的核心是物品画像和用户画像的构建。其中物品的画像，即物品的标签，一般是通过人工的手段来完成，如音乐的流派标签（民谣、电子、嘻哈等），电影的风格标签（科幻、动作、战争等）等。

CB算法的难点是用户画像的建设，即用户信息的标签化，它是以用户行为数据作为基础，抽象出一个用户的信息全貌，包括与用户相关的基础属性、社交属性等。用户画像的构建具有领域性，即不同的场景，不同的平台下构建的画像维度不一致。图11.16展示了用户画像的几个不同维度。



图11.16 用户画像

当完成了物品画像和用户画像的建设之后，可以对标签进行匹配推荐。匹配的算法有很多，下面介绍一种基于TF-IDF思想的匹配算法。

设用户 $u$ 的标签向量为 $\{u_1, u_2, \dots, u_n\}$ ，物品 $i$ 的标签向量为 $\{i_1, i_2, \dots, i_n\}$ ， $u_k$ 是指用户 $u$ 对标签 $k$ 的偏好度，值越大表示用户越偏好，同理， $i_k$ 是指物品 $i$ 对标签 $k$ 的符合程度，值越大，则置信度越高，简单的匹配公式可以表示为：

$$s(u, i) = \sum_k u_k \times i_k \tag{11.61}$$

但这个计算公式在实际使用时，会遇到一个严重的问题，就是对于热门的标签权重很高，导致热门物品被经常推送给用户，无法解决推荐的长尾问题。为此，可以借鉴TF-IDF的思想，对热门的标签做出惩罚，修正后的得分公式为：

$$s(u, i) = \sum_k \frac{u_k \times i_k}{\ln(1 + b_k)} \tag{11.62}$$

其中， $b_k$ 表示了标签 $k$ 被多少个不同的用户使用过， $b_k$ 可以反映出标签的热度。同理，也可以对物品的标签热度进行惩罚，设 $c_k$ 表示了标签 $k$ 这个标签已经被打在多少个物品上，式(11.61)可以进一步修改为：

$$s(u, i) = \sum_k \left( \frac{u_k}{\ln(1 + b_k)} \times \frac{i_k}{\ln(1 + c_k)} \right) \tag{11.63}$$

**协同过滤 (Collaborative Filtering Recommendations, 简称CF算法)：**是个性化推荐中应用最广泛的一种推荐算法。CF算法的思想可以用一句话来简单概括，即“物以类聚，人以群分”，用户通过不断地和网站互动产生相互影响、相互协助的行为流水，并利用这种流水数据找到与自己兴趣相符的物品。常用的CF算法可以分为基于邻域的协同过滤和基于隐特征的协同过滤。本节介绍以user-based和item-based为代表的邻域协同过滤算法，下一节将重新回到RBM，介绍如何利用RBM构建隐特征协同过滤模型。

基于邻域的协同过滤首先将用户的行为流水数据抽象为一个矩阵，行代表用户，列代表物品，如图11.17所示。

	item1	item2	item3	item4	item5
user1	*			*	
user2	*		*		*
user3		*		*	*
user4		*	*		
user5	*			*	

图11.17 用户行为流水矩阵

用户的行为数据一般可以分为显性数据 (explicit feedback) 和隐性数据 (implicit feedback), 显性行为数据是那些由用户明确表示对物品是否喜欢的行为, 比如在音乐 App 中, 用户对歌曲的收藏、删除等操作; 在视频网站中, 用户对视频的“赞”和“踩”, 图11.18展示了几种常见的显性反馈形态。



图11.18 用户的显性行为

与显性行为相比, 用户绝大部分的行为数据都属于隐性行为数据, 诸如浏览网页, 普通的听歌操作等, 这些行为一般不能轻易判断用户是否喜欢某一种物品, 可以通过停留时长、商品的画像、用户画像等基础数据综合考虑。

利用显性行为数据和隐性行为数据对图11.17的矩阵 $\mathbf{W}$ 进行填充, 最简单的方法是0-1填充。若用户 $u$ 喜欢物品 $i$ , 则 $\mathbf{W}[u, i] = 1$ ; 否则其他情况下,  $\mathbf{W}[u, i] = 0$ 。使用0-1方式填充, 可以采用Jaccard相似度来求解相似用户或者相似物品, Jaccard相似度的定义为:

$$\text{Jaccard}(u, v) = \frac{N(u) \cap N(v)}{N(u) \cup N(v)} \quad (11.64)$$

其中 $N(u)$ 表示用户 $u$ 的喜欢物品列表, 以图11.19为例, 用户 $A$ 的喜爱列表为 $\{a, b, c\}$ , 用户 $B$ 的喜爱列表为 $\{a, d\}$ , 故有:

$$\text{Jaccard}(A, B) = \frac{\{a, b, c\} \cap \{a, d\}}{\{a, b, c\} \cup \{a, d\}} = \frac{1}{4}$$

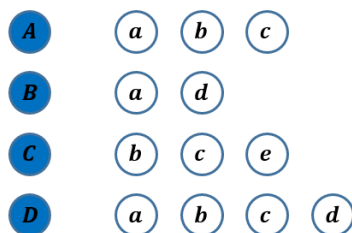


图11.19 用户的行为数据列表，用户数据集为 $\{A, B, C, D\}$ ，物品的数据集为 $\{a, b, c, d, e\}$

如果矩阵采用浮点数或者整数值填充，那么求取用户或物品之间的相似度可以采用余弦相似度、Pearson相关系数等指标。

基于邻域的协同过滤会存在三个缺点。

第一，无法解决冷启动问题。事实上，所有的协同过滤算法都没有办法解决冷启动问题，解决这一类问题常用的方法是试错法，也就是通过不断试错，把用户的喜好迅速限制在一个收敛的空间范围内。此外，基于内容的推荐也是解决冷启动物品推荐的常用手段。

第二，数据的稀疏。在实际的产品应用中，用户行为矩阵是一个极度稀疏的矩阵，要解决这个问题，最直接的方法是，结合用户画像数据来填充，或者对矩阵进行二级甚至三级扩展。用户的评分矩阵可以抽象为一个二分图，如图11.20所示，只有有评分数据，用户 $u$ 与物品 $i$ 之间才会有边相连，也就是说从用户 $u$ 到物品 $i$ 的距离为1，二级扩展是指寻找距离为3的路径来扩充，用户 $A$ 对物品 $c$ 没有评分数据，但可以通过 $(A \rightarrow b) \rightarrow (b \rightarrow B) \rightarrow (B \rightarrow c)$ 的评分路径来扩展得到用户 $A$ 对物品 $c$ 的评分。

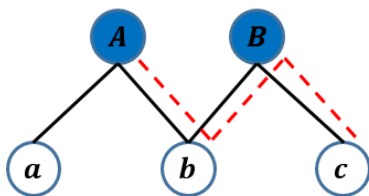


图11.20 评分的二级扩展

第三，容易造成热门的物品被频繁推送。这主要是由于热门的物品关注的用户数太多，造成在计算相似度时，与很多其他物品的相似分数都很高，因此还需要对热门的物品进行权重的打压。



### 11.5.3 RBM与协同过滤

如前所述，基于邻域的协同过滤是直接根据用户的行为流水，为用户推荐与其爱好相近的物品，除了上节提到的3个缺点外，邻域协同算法还存在一个不足，那就是该算法只能挖掘到表面浅层的关系。考察如图11.21的例子，把用户的行为流水抽象为一个矩阵，行表示用户，列代表物品，考察item1和item2，利用Jaccard相似度式(11.63)求解item1和item2的相似性，由于item1和item2的行没有任何相关性，因此，当用户的行为爱好中含有item1时，很可能永远不会给该用户推送item2。

	item1	item2	item3	item4	item5
user1	*			*	
user2	*		*		*
user3		*		*	*
user4		*	*		
user5	*			*	

图11.21 邻域协同无法找到item1和item2的关联性

但这并不能说明item1和item2之间完全不相关，事实上，可以考察item1、item2和item3三者之间的关系，如图11.22所示，以item3作为隐秘的纽带，可以得知它们的相似度（采用Jaccard相似度）为：

$$\text{similarity}(\text{item1}, \text{item3}) = \frac{1}{4}, \quad \text{similarity}(\text{item2}, \text{item3}) = \frac{1}{3}$$

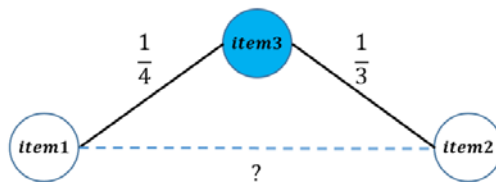


图11.22 考察item1、item2和item3的相关性

因此，有理由相信item1和item2之间存在着一定的关联性，而这种关联性不能通过显式的协同过滤得到。RBM模型是具有隐特征提取功能的无向概率图模型。下面考察如何将RBM作用于协同推荐场景中。

以MovieLens数据集为例来解析RBM在协同过滤推荐中的应用。MovieLens创建于1997年，是一个推荐系统和虚拟社区网站，其主要功能是应用协同过滤技术来预测用

户对电影的喜好，由隶属于美国明尼苏达大学计算机系的GroupLens Research实验室负责维护。MovieLens根据用户对一部分电影的评分，预测该用户对其他电影的评分。当一个新的用户进入MovieLens，他首先需对15部电影评分，评分的范围从1~5分。MovieLens按照评分数据的大小划分了不同的数据集，在本书将采用1M的数据集，该数据集含有来自超过6千名用户对4千部电影的评分记录，一共有100万条评分数据。

首先对训练数据集进行一个统计，图11.23展示了用户和电影数据的一些统计信息：左图是用户的评分记录分布，每一个用户至少对20部电影有评分。右图是电影的评分分布，每一部电影都至少被一个用户评分。

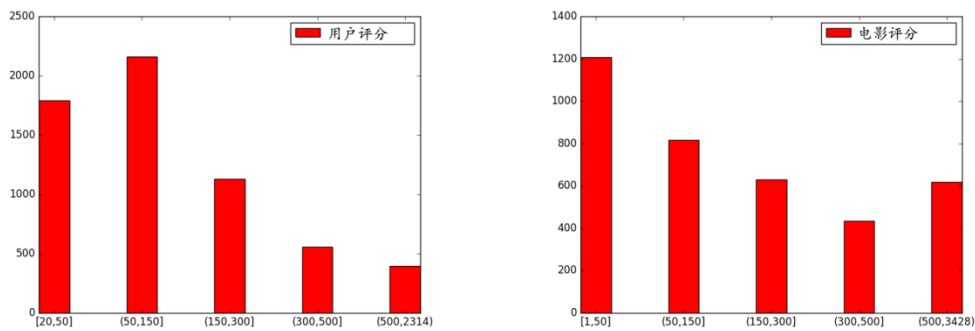


图11.23 MovieLens 1M数据集统计分布

利用RBM来进行协同过滤推荐，主要通过两个步骤来进行。

**第1步：训练权重参数。**与11.4节提到的学习过程类似，RBM通过将用户的评分数据作为训练集，采用对比散度算法来训练模型的权重参数。但传统的RBM模型在输入层中，每一个神经元都是一个二值神经元，而推荐系统中，输入的数据是用户对物品的评分数据。前面我们提到，MovieLens的用户评分是从1~5分，RBM模型无法使用一个神经元来表示用户对物品的评分，为此对RBM模型进行扩展，如图11.24所示，输入层的每一个二值神经元扩展为一个由5个二值神经单元组成的列表。如果用户对物品1的评分为3，那么对应的物品1的可视层向量为： $(v_1^1 = 0, v_1^2 = 0, v_1^3 = 1, v_1^4 = 0, v_1^5 = 0)$ 。

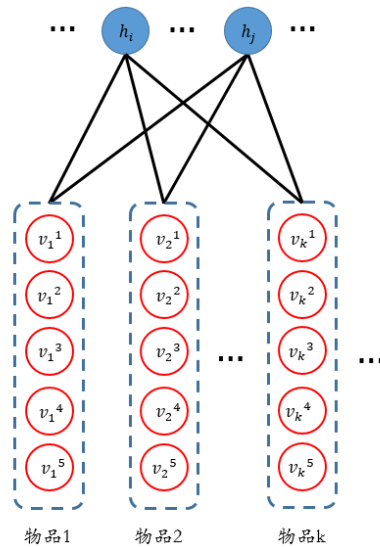


图11.24 将RBM模型扩展应用于协同过滤推荐中

事实上，可以继续把图11.24的可视层进行水平扩展，得到图11.25的结构，这样就将推荐问题转换为标准的RBM学习问题。

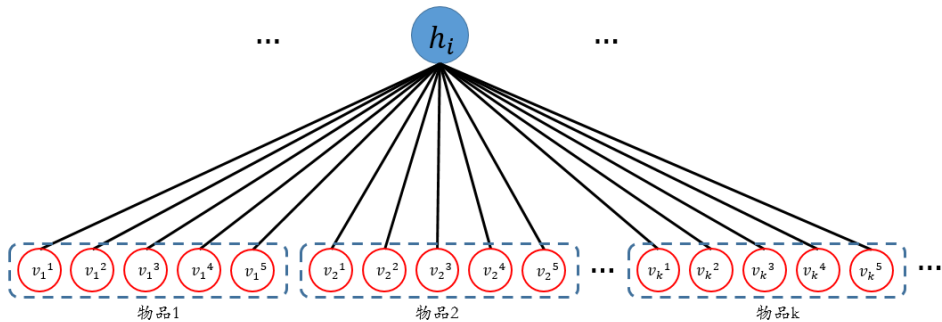


图11.25 将图11.24扩展后得到的标准RBM模型

每一个用户的训练数据构成了一个单独的RBM子模型，在训练过程中，输入层仅考虑有评分的物品，忽略没有评分的物品，如图11.26所示。

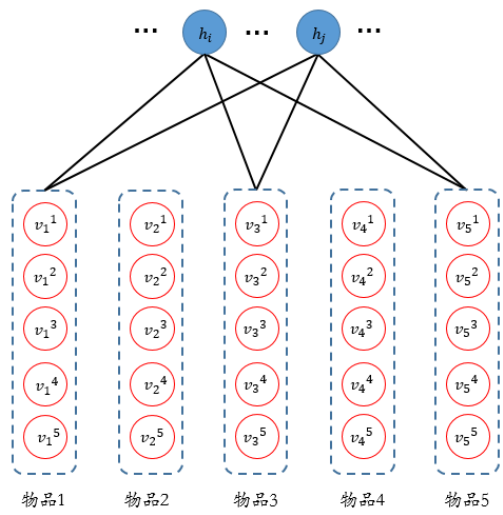


图11.26 某一个用户只对物品1、物品3和物品5有操作，则RBM只需要将对应的  
可视层神经元与隐藏层神经元相连

**第2步：重构输入层。**经过第一步的操作，采用对比散度算法对神经网络的参数训练完毕，下面利用网络来重构训练数据。重构过程由编码和解码两个阶段构成。

**编码过程：**利用原始输入数据，求取隐藏层的隐向量表示 $p(h|v) = (p(h_1|v), p(h_2|v), \dots, p(h_m|v))$ 。

其中 $p(h_j|v)$ 是一个条件推断求解过程，由式(11.31)和式(11.32)可以求解。

**解码过程：**与编码过程相反，利用上一步求得的隐藏层向量，反向重构可视层的表示。由于扩展的RBM模型中，输入层每一个物品由5个子神经元构成，为了重构用户对每一个物品的喜好，可以对输出数据构建softmax模型，如式(11.65)所示：

$$p(v_i^k = 1|h) = \frac{\exp\left(\left(\sum_{j=1}^m w_{ij}^k \times h_j\right) + a_i^k\right)}{\sum_{t=1}^5 \exp\left(\left(\sum_{j=1}^m w_{ij}^t \times h_j\right) + a_i^t\right)} \tag{11.65}$$

与学习过程和编码过程不同，重构的解码过程将考虑所有的可视层神经元，这样不仅可以重新还原已评分的物品数据，还可以对没有任何操作的数据进行预测，推荐的结果可以通过对未知物品的评分，从高到低进行排序，推送给用户。图11.27展示了通过对MovieLen数据集进行重构后的效果图。

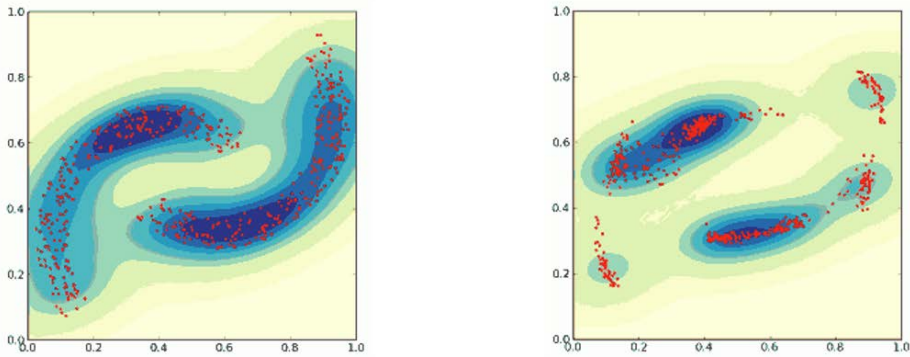


图11.27 左图为MovieLen的训练数据集，右图为重构后的数据

推荐算法的离线效果评价标准有很多，本文采用常见的RMSE来考察算法的效果，RMSE的全称是Root Mean Square Error，即均方根误差，也称为标准误差，其计算公式为：

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (v_i - pv_i)^2} \quad (11.66)$$

其中， $v_i$ 表示原始值， $pv_i$ 表示预测值，RMSE可以反映出数据的预测准确度，对1M数据集分别采用了SVD、RBM两个模型来进行对比，结果如下图11.28所示。

	参数设置	RMSE
SVD	$num\_iter = 200, reg = 0.05, learning\_rate = 0.005, C = 50$ 其中 $C$ 表示降维后的特征数	0.92375
RBM	$num\_iter = 200, cd\_k = 1, m = 50$ 其中 $m$ 表示隐藏层的节点数， $cd\_k$ 是求解负梯度时的采样步数	0.92039

图11.28 SVD和RBM的结果对比图

SVD和RBM是两个常用的隐协同过滤模型，从结果可以看出，两者的效果相当。注意，这里面的结果并不是模型的最优值，读者可以通过自行调整参数值来获得更优的解。此外，RBM模型还有其他的扩展模型，可以进一步提升预测的准确度，如Conditional RBM、Conditional Factored RBM等，读者可参考文献[6]，这里不再详述。

## 参考文献：

---

- [1] Daphne Koller, Nir Friedman. Probabilistic Graphical Models, Principles and Techniques. The MIT Press. ISBN-13: 978-0262013192.
- [2] Ackley, David H; Hinton Geoffrey E; Sejnowski, Terrence J (1985). "A learning algorithm for Boltzmann machines", Cognitive science, Elsevier, 9 (1): 147–169.
- [3] Y LeCun. A Tutorial on Energy-Based Learning. MIT Press, 2006.
- [4] Hinton, G. E. (2002). Training Products of Experts by Minimizing Contrastive Divergence. Neural Computation. 14 (8): 1771–1800.
- [5] Geoffrey Hinton (2010). A Practical Guide to Training Restricted Boltzmann Machines. UTM TR 2010–003, University of Toronto.
- [6] Ruslan Salakhutdinov, Andriy Mnih, Geoffrey Hinton. Restricted Boltzmann Machines for Collaborative Filtering. International Conference of machine learning. ICML 2007.
- [7] Francesco Ricci, Lior Rokach, Bracha Shapira. Recommender Systems Handbook, Springer; 2nd ed. 2015 edition. ISBN-13: 978-1489976369.

# 12

## 递归神经网络

递归神经网络是两类人工神经网络的总称,分别是**时间递归神经网络 (Recurrent Neural Network)**和**结构递归神经网络 (Recursive Neural Network)**(注:也有很多文献分别称之为递归神经网络和循环神经网络)。

在前面几章讲解的网络模型中,预先假设训练数据之间是相互独立的关系,但在很多实际的应用中,数据之间是相互依赖的,例如在时间序列相关的输入场景下,信息之间的传递更多的是一种相互传承的关系。在现实生活中,我们遇到的很多例子都属于这种情况,例如,当我们思考问题的时候,我们都是已有的经验和知识的基础上,根据当前的实际情况来综合考虑问题,而不会把过往的经验和记忆丢弃。在空间结构场景中,数据之间存在空间的组合关系,可以将整体数据划分为局部的小结构,由局部的小结构推导出整体的性质。

本章将对递归神经网络(注意,为了区分时间递归网络和结构递归网络,下面我们使用RNN时,指代的是时间递归网络)的结构和原理进行详细讲解,具体包括下面五大部分内容。

**12.1节: RNN的相关基础知识。**本节将对传统的Elman递归神经网络的结构进行详细讲解。

**12.2节: BPTT与梯度消失。**首先讲解如何使用时间反向传播算法来训练递归神经网络,然后详细分析BPTT为何会导致梯度消失的问题。

12.3节：长短时记忆网络。长短时记忆网络，即LSTM，是当前应用最广泛的一种RNN变种模型。传统的Elman模型，由于其隐藏单元的设计只有简单的激活操作，导致模型的训练存在梯度消失的缺点，无法有效获取前面较长时间的记忆信息，LSTM就是为了解决这个问题而提出的一种网络结构，我们将在本节详细讲解LSTM的设计原理。此外，还将简要介绍LSTM的另一种变形网络：门控递归单元网络，即GRU，GRU可以看成是在LSTM的基础上简化了隐藏层的逻辑设计。

12.4节：讲解一种新型的网络——结构递归网络，结构递归网络被广泛应用于构造句子的向量表示，将对结构递归网络的原理和训练方法进行深入的探讨。

12.5节，将通过语言模型的例子，来讲解RNN在解决自然语言处理等问题上的具体应用。

## 12.1 Elman递归神经网络

当前，RNN在基础研究领域和工程领域都取得了很多突破性进展。例如，在自然语言处理领域，Bengio提出在构建语言模型时，可以采用神经网络模型<sup>[1]</sup>来改进传统的N元统计模型（N-gram Model）。此外，RNN也被广泛应用于机器翻译领域<sup>[2]</sup>、语音识别<sup>[3]</sup>和个性化推荐领域<sup>[4]</sup>。

首先来回顾第8章前馈神经网络的相关知识。前馈神经网络将信息从输入层，经多个隐藏层处理，最后到达输出层，数据的流动是单向传递的过程，而RNN处理的对象是一种时间序列数据，它将数据信息流以一种循环的方式进行传递处理。

RNN具有的两个特点。

- **持续性**：在时间序列信息中，前后数据间不是相互独立，而是相互依赖的，当前阶段的输出结果受到过去的决策影响，同理，当前阶段的输出也会影响到后面的决策。

- **记忆性**：RNN可以保留序列的“记忆”信息。例如，在序列式的个性化推荐场景中，为了在当前时刻给用户选择合适的推送数据，需要保留用户过去的操作点击行为，我们称前面的这些用户行为记录为“记忆”，类似于人的大脑机制。这些“记忆”将有助于对信息进行筛选，以音乐推荐为例，RNN为我们保留了过去的用户点击行为，这些“记忆”的积累信息可以让推荐系统捕获用户在过



去一段时间内的口味和情感变化，如流派、歌手、语言喜好等。

图12.1是传统的Elman RNN网络模型的展开效果图，从图中可以看到，通过展开操作，可以从理论上把网络模型扩展为无限维，也就是无限的时间序列。

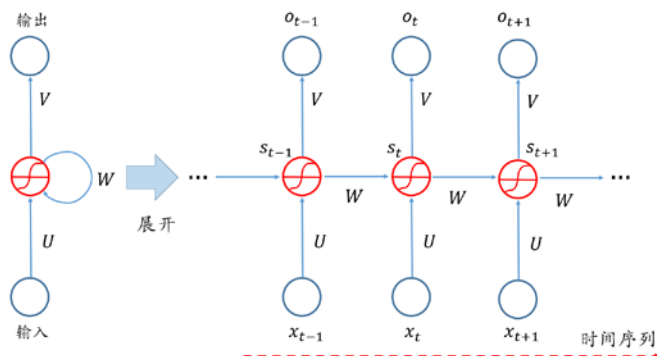


图12.1 将RNN模型展开后的效果图

RNN在每一时间步 $t$ 有相同的网络结构，Elman RNN一共有3个参数值，分别是 $U$ 、 $V$ 和 $W$ 。设输入层的神经元个数为 $n$ ，隐藏层的神经元个数为 $m$ ，输出层的神经元个数为 $r$ ，则 $U$ 是连接输入层和隐藏层的权重矩阵，大小为 $(n \times m)$ 维； $W$ 是连接上一时间步的隐藏层单元与当前时间步的隐藏层单元的权重矩阵，大小为 $(m \times m)$ 维； $V$ 是连接隐藏层单元与输出层单元的权重矩阵，大小为 $(m \times r)$ 维。

本节接下来的部分将对Elman网络的结构与符号进行形式化定义。图12.2展示了Elman RNN模型在每一个时间步的网络结构。

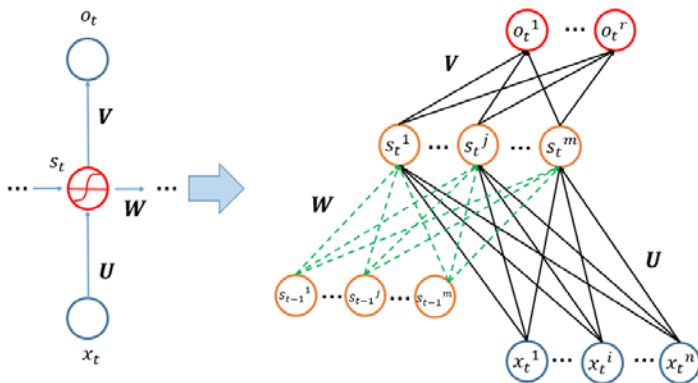


图12.2 Elman RNN模型在每一个时间步的网络结构图

- 最下方是RNN的输入层，用 $X$ 来表示， $X = (x_0, x_1, \dots, x_T)$ ，其中， $x_t$ 表示序列

中的第 $t$ 时刻或第 $t$ 时间步的输入数据， $\mathbf{x}_t$ 通常是一个向量，记为 $\mathbf{x}_t = (\mathbf{x}_t^1, \mathbf{x}_t^2, \dots, \mathbf{x}_t^n)^T$ 。

- 中间部分是网络的隐藏层，记为 $S = (s_0, s_1, \dots, s_T)$ 。与输入层一样， $\mathbf{s}_t$ 同样是一个向量，记为 $\mathbf{s}_t = (\mathbf{s}_t^1, \mathbf{s}_t^2, \dots, \mathbf{s}_t^m)^T$ ， $m$ 为隐藏层向量的维度。隐藏层是RNN模型中最核心的部分，也是RNN处理记忆信息的地方，先用数学语言来描述记忆信息的处理过程：

$$\mathbf{s}_t = \begin{cases} 0, & t = -1 \\ \sigma(\mathbf{s}_{t-1}, \mathbf{x}_t), & \text{其他} \end{cases} \tag{12.1}$$

其中，函数 $\sigma$ 是一个非线性的、平滑的、有界的激活函数，可以取sigmoid、tanh、ReLU等。通常来说，需要设定一个特殊的初始隐藏单元 $\mathbf{s}_{-1}$ ，也就是初始的记忆状态，一般情况下，会将其初始化为0向量，从式(12.1)可以看出，第 $t$ 时间步的记忆信息由前面 $(t - 1)$ 个时间步的记忆结果 $\mathbf{s}_{t-1}$ 和当前的输入 $\mathbf{x}_t$ 共同决定，这些记忆信息保存在隐藏层中，不断向后传递，跨越多个时间步，影响每一个新输入数据的处理。

RNN的这种循环处理信息的机制，与人类大脑处理记忆的过程颇为相似，人类记忆会在大脑中不断进行循环更新，并逐渐沉淀下来，成为日常生活中的经验知识。在后面的章节中，我们将会发现，不同类型的递归网络模型，本质上是隐藏层设计的不同，在Elman递归模型中，其隐藏层神经元设计较为简单，它把当前的输入数据和传递过来的记忆数据进行线性组合后，利用非线性激活函数进行压缩，激活函数可以采用第 8.5.2 节提到的激活函数，它们所具有的性质在RNN模型中同样适用。这里取激活函数为sigmoid函数，则式(12.1)可以转化为：

$$\mathbf{s}_t = \begin{cases} 0, & t = -1 \\ \text{sigmoid}(\mathbf{U}^T \times \mathbf{x}_t + \mathbf{W}^T \times \mathbf{s}_{t-1}), & \text{其他} \end{cases} \tag{12.2}$$

其中 $\mathbf{U}$ 和 $\mathbf{W}$ 是网络的其中两个参数，隐藏层神经元结构可以用图12.3来可视化表示。

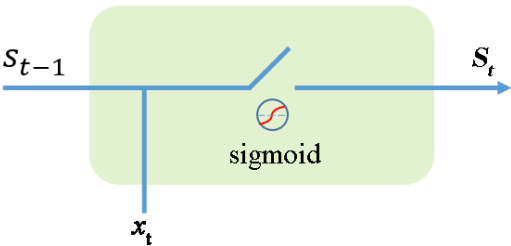


图12.3 Elman隐藏层设计的效果图

- RNN网络的最后一层结构是最上方的输出层，用 $O$ 来表示，记为：

$$O = (o_0, o_1, \dots, o_T)$$

$o_t$ 表示序列在第 $t$ 步的输出结果，与输入层和隐藏层不同，输出层的设计则要灵活很多，并没有规定每一时间步都必须有输出。例如，在文本的情感分类中，关心的是整个句子的情感结果，而不是每一个单词，因此只需要在最后时间步有输出层即可。对于分类模型，输出层的结果一般情况下仅依赖于当前的隐藏层数据，即满足：

$$o_t = \text{softmax}(\mathbf{V}^T \times \mathbf{s}_t) \quad (12.3)$$

## 12.2 时间反向传播

BPTT, 全称为BackPropagation Through Time, 即时间反向传播, 是训练RNN网络模型的标准算法。它的流程与反向传播算法很相似。回顾Elman网络模型的计算流程, 在任意的第 $t$ 时间步, 分别得到隐藏层和输出层的结果如下所示:

$$\mathbf{s}_t = \text{sigmoid}(\mathbf{U}^T \times \mathbf{x}_t + \mathbf{W}^T \times \mathbf{s}_{t-1}) \quad (12.4)$$

$$\mathbf{f}_t = \text{softmax}(\mathbf{V}^T \times \mathbf{s}_t) \quad (12.5)$$

为了简单起见, 没有添加偏移量参数, 但不会影响后面的推导流程和结论的输出。我们的目标函数是计算每一时间步的输出层数据的交叉熵, 例如, 在第 $t$ 层, 其对数损失函数可表示为:

$$L(\mathbf{f}_t, y_t) = - \sum_i 1_{(y_t=i)} \ln(\mathbf{f}_t)_i = - \ln(\mathbf{f}_t)_{y_t} \quad (12.6)$$

其中 $\mathbf{f}_t$ 是预测输出向量,  $(\mathbf{f}_t)_i$ 表示预测值为 $i$ 的概率,  $y_t$ 是真实输出, 其中 $1_{(y_t=i)}$ 称为indicator function, 满足:

$$1_{(y_t=i)} = \begin{cases} 1, & y_t = i \\ 0, & y_t \neq i \end{cases} \quad (12.7)$$

在RNN模型的训练中, 全部时间序列构成一个训练样本数据, 因此, 对于单个数据集, 其对应的损失函数为:

$$L(f, y) = \sum_t L(\mathbf{f}_t, y_t) = - \sum_t \ln(\mathbf{f}_t)_{y_t} \quad (12.8)$$

与反向传播算法类似, BPTT算法的核心是求解参数的导数, 然后利用梯度下降等最优化方法来进行参数的迭代更新。为了求解式(12.8)的参数导数, 首先考察每一

时间步的损失 $L(\mathbf{f}_t, y_t)$ 。由于和的导数等于导数的和，当 $L(\mathbf{f}_t, y_t)$ 的结果已知，根据 $t$ 的任意性，可以求解出 $L(\mathbf{f}, y)$ 。下面只讨论第 $t$ 时间步的损失 $L(\mathbf{f}_t, y_t)$ 的求导过程：

$$\frac{\partial L(\mathbf{f}_t, y_t)}{\partial \theta} = \frac{\partial -\ln(\mathbf{f}_t)_{y_t}}{\partial \theta}, \quad \theta = \mathbf{U}, \mathbf{V}, \mathbf{W} \quad (12.9)$$

• 当 $\theta = \mathbf{V}$ 时，这是最简单的情况，把(12.5)式代入(12.9)式可得：

$$\begin{aligned} \frac{\partial -\ln(\mathbf{f}_t)_{y_t}}{\partial V_{ij}} &= \frac{-1}{(\mathbf{f}_t)_{y_t}} \times \frac{\partial(\mathbf{f}_t)_{y_t}}{\partial V_{ij}} \\ &= \frac{-1}{(\mathbf{f}_t)_{y_t}} \times \frac{\partial}{\partial V_{ij}} \left( \frac{\exp(V_{*,y_t}^T \times s_t)}{\sum_{y=1}^r \exp(V_{*,y}^T \times s_t)} \right) \\ &= \frac{-1}{(\mathbf{f}_t)_{y_t}} \times \frac{\frac{\partial \exp(V_{*,y_t}^T \times s_t)}{\partial V_{ij}} \times \sum_{y=1}^r \exp(V_{*,y}^T \times s_t) - \exp(V_{*,y_t}^T \times s_t) \times \frac{\partial \sum_{y=1}^r \exp(V_{*,y}^T \times s_t)}{\partial V_{ij}}}{\left( \sum_{y=1}^r \exp(V_{*,y}^T \times s_t) \right)^2} \\ &= \frac{-1}{(\mathbf{f}_t)_{y_t}} \times \left( \frac{1_{(y_t=j)} \times \exp(V_{*,y_t}^T \times s_t) \times (s_t)_i \times \sum_{y=1}^r \exp(V_{*,y}^T \times s_t)}{\left( \sum_{y=1}^r \exp(V_{*,y}^T \times s_t) \right)^2} \right. \\ &\quad \left. - \frac{\exp(V_{*,y_t}^T \times s_t) \times \exp(V_{*,j}^T \times s_t) \times (s_t)_i}{\left( \sum_{y=1}^r \exp(V_{*,y}^T \times s_t) \right)^2} \right) \\ &= \frac{-1}{(\mathbf{f}_t)_{y_t}} \times (1_{(y_t=j)} \times (\mathbf{f}_t)_{y_t} \times (s_t)_i - (\mathbf{f}_t)_j \times (\mathbf{f}_t)_{y_t} \times (s_t)_i) \\ &= -(1_{(y_t=j)} - (\mathbf{f}_t)_j) \times (s_t)_i \end{aligned}$$

故有：

$$\frac{\partial -\ln(\mathbf{f}_t)_{y_t}}{\partial V_{ij}} = -(1_{(y_t=j)} - (\mathbf{f}_t)_j) \times (s_t)_i \quad (12.10)$$

成立。对于矩阵 $\mathbf{V}$ ，可以把式(12.10)写成：

$$\frac{\partial -\ln(\mathbf{f}_t)_{y_t}}{\partial \mathbf{V}} = s_t \times (\mathbf{f}_t - e(y_t))^T \quad (12.11)$$

其中：

$$e(y_t) = (1_{(y_t=1)}, 1_{(y_t=2)}, \dots, 1_{(y_t=r)})^T$$

$$\mathbf{f}_t = ((f_t)_1, (f_t)_2, \dots, (f_t)_r)^T$$

$$\mathbf{s}_t = ((s_t)_1, (s_t)_2, \dots, (s_t)_m)^T$$

验证维度的正确性,  $\mathbf{s}_t \in \mathbf{R}^{m \times 1}$ ,  $\mathbf{f}_t - \mathbf{e}(y_t) \in \mathbf{R}^{r \times 1}$ , 故式(12.11)的结果为  $m \times r$  的矩阵。从结果可以看出, 对  $\mathbf{V}$  的求导仅仅由  $\mathbf{s}_t$  和  $\mathbf{f}_t$  构成, 因此其结果并不会随着时间的推进而发生梯度消失的情况。

• 当  $\theta = \mathbf{W}$  时, 把式(12.4)代入式(12.9)可得:

$$\frac{\partial -\ln(\mathbf{f}_t)_{y_t}}{\partial W_{ij}} = \frac{-1}{(\mathbf{f}_t)_{y_t}} \times \frac{\partial (\mathbf{f}_t)_{y_t}}{\partial s_t} \times \frac{\partial s_t}{\partial W_{ij}} \quad (12.12)$$

式(12.12)主要由两部分求导公式构成, 其中  $\frac{\partial (\mathbf{f}_t)_{y_t}}{\partial s_t}$  可由对式(12.5)求导得到:

$$\begin{aligned} \frac{\partial (\mathbf{f}_t)_{y_t}}{\partial s_t} &= \frac{\partial}{\partial s_t} \left( \frac{\exp(V_{*,y_t}^T \times s_t)}{\sum_{y=1}^r \exp(V_{*,y}^T \times s_t)} \right) \\ &= \frac{\frac{\partial \exp(V_{*,y_t}^T \times s_t)}{\partial s_t} \times \sum_{y=1}^r \exp(V_{*,y}^T \times s_t) - \exp(V_{*,y_t}^T \times s_t) * \frac{\partial \sum_{y=1}^r \exp(V_{*,y}^T \times s_t)}{\partial s_t}}{\left( \sum_{y=1}^r \exp(V_{*,y}^T \times s_t) \right)^2} \\ &= \left( \frac{\exp(V_{*,y_t}^T \times s_t) \times V_{*,y_t}^T \times \sum_{y=1}^r \exp(V_{*,y}^T \times s_t)}{\left( \sum_{y=1}^r \exp(V_{*,y}^T \times s_t) \right)^2} \right. \\ &\quad \left. - \frac{\exp(V_{*,y_t}^T \times s_t) \times \sum_{y=1}^r (V_{*,y}^T \times \exp(V_{*,y}^T \times s_t))}{\left( \sum_{y=1}^r \exp(V_{*,y}^T \times s_t) \right)^2} \right) \\ &= \left( (\mathbf{f}_t)_{y_t} \times V_{*,y_t}^T - (\mathbf{f}_t)_{y_t} \times \sum_{y=1}^r (V_{*,y}^T \times (\mathbf{f}_t)_y) \right) \end{aligned}$$

整理归纳上式得:

$$\frac{\partial (\mathbf{f}_t)_{y_t}}{\partial s_t} = \left( (\mathbf{f}_t)_{y_t} \times V_{*,y_t}^T - (\mathbf{f}_t)_{y_t} \times \sum_{y=1}^r (V_{*,y}^T \times (\mathbf{f}_t)_y) \right) \quad (12.13)$$

继续考察  $\frac{\partial s_t}{\partial W_{ij}}$  的求导结果, 由于  $\mathbf{s}_t$  包含了  $\mathbf{s}_{t-1}$ , 同理,  $\mathbf{s}_{t-1}$  包含了  $\mathbf{s}_{t-2}$ , 而每一个时间步都采用相同的参数  $\mathbf{W}$ , 因此利用链式法则进一步将式(12.12)化简:

$$\begin{aligned} \frac{\partial -\ln(\mathbf{f}_t)_{y_t}}{\partial W_{ij}} &= \frac{-1}{(\mathbf{f}_t)_{y_t}} \times \frac{\partial (\mathbf{f}_t)_{y_t}}{\partial s_t} \times \frac{\partial s_t}{\partial W_{ij}} \\ &= \frac{-1}{(\mathbf{f}_t)_{y_t}} \times \frac{\partial (\mathbf{f}_t)_{y_t}}{\partial s_t} \times \left( \sum_{i=0}^t \left( \frac{\partial s_t}{\partial s_i} \times \frac{\partial s_i}{\partial W_{ij}} \right) \right) \quad (12.14) \end{aligned}$$

其中 $\frac{\partial s_t}{\partial s_i}$ 由下面的公式求得：

$$\frac{\partial s_t}{\partial s_i} = \prod_{j=i}^{t-1} \frac{\partial s_{j+1}}{\partial s_j} \quad (12.15)$$

把式(12.15)代入式(12.14)，得到最终的参数求导公式为：

$$\frac{\partial -\ln(f_t)_{y_t}}{\partial W_{ij}} = \frac{-1}{(f_t)_{y_t}} \times \frac{\partial (f_t)_{y_t}}{\partial s_t} \times \left( \sum_{i=0}^t \left( \left( \prod_{j=i}^{t-1} \frac{\partial s_{j+1}}{\partial s_j} \right) \times \frac{\partial s_i}{\partial W_{ij}} \right) \right) \quad (12.16)$$

令 $E_t = -\ln(f_t)_{y_t}$ ，则 $E_t$ 表示第 $t$ 时间步的损失，图12.4展示了这种损失误差在时间维度上的传递过程。

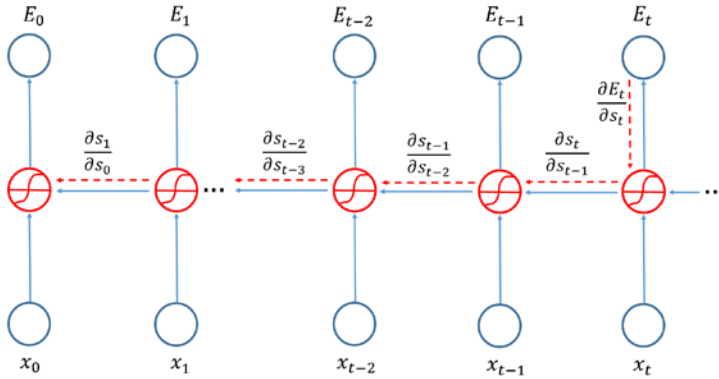


图12.4 BPTT随时间向前传递的效果图

考察式(12.16)，公式中存在导数传递子项，即：

$$\prod_{j=i}^{t-1} \frac{\partial s_{j+1}}{\partial s_j}$$

$s_{j+1}$ 对 $s_j$ 的求导结果是一个含有sigmoid函数的导数，sigmoid函数的导数的表示形式为：

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

sigmoid函数的导数的值域在范围 $[0, \frac{1}{4}]$ 内，因此，每一次反向传递的过程中，梯度都会以 $\frac{1}{4}$ 的速度递减，可以想象，梯度的计算是随着往前传递时间步数的增加而呈现出指数级的递减趋势，离当前时间越远，梯度减少越明显。例如假设当前时刻为 $t$ ，则在 $(t-3)$ 时刻，其梯度将减少至少 $\left(\frac{1}{4}\right)^3 = \frac{1}{64}$ 。

这种现象称之为**梯度消失 (vanishing gradient)**，与BP算法一样，利用BPTT求解递归神经网络之所以会导致梯度消失，主要是由于激活函数的导数所引发的梯度下降叠加。在RNN网络中，梯度消失导致的后果是模型无法保留前面较远时间的记忆，因此，在实际的应用中效果不甚理想。在自然语言处理中，当一个单词能表达多种不同的含义时，为了确定其准确的意义，不仅要考察其周围的单词，很可能还需要考察上下文环境，比如更早的句子信息，甚至前一段的信息。在这种关系跨度较大的应用中，Elman递归网络的效果并不理想，后面将讲解另外两种改进的RNN模型，它们有效克服了梯度消失的缺点，对于捕获长时间的记忆信息效果非常显著。

## 12.3 长短时记忆网络

长短时记忆网络 (Long Short Term Memory, 简称 LSTM)，是时间递归神经网络的一种变种形态，由Hochreiter和Schmidhuber在1997年提出<sup>[5]</sup>，目前已经被广泛应用于机器翻译、语音识别等自然语言处理领域。传统的Elman模型，由于存在梯度消失等缺点，因此不能有效地保留长时间的记忆信息。LSTM则对隐藏层的设计做了相应改进。LSTM的整体结构与图12.1基本一致，但隐藏层的设计更复杂，并有效克服了梯度消失问题，与Elman只有激活层的结构不同，LSTM隐藏层的核心设计，是一种叫记忆体 (cell state) 的信息流，它负责把记忆信息从序列的初始位置传递到序列的末端，并通过4个相互交互的“门”单元，来控制着在每一时间步 $t$ 对记忆信息值的修改。

### 1. 忘记门

忘记门 (Forget Gate Layer)，其作用是控制着要从前面的记忆中丢弃多少信息。以音乐个性化推荐为例，用户过去的行为操作，对当前的推荐决策有重要的影响。对于正向行为，如用户对某一位歌手或者某一种流派的歌曲特别感兴趣，那么这种正向操作的“记忆”将得到加强；相反，对于负向的操作行为，如删除、跳过等行为，其对应的信息则会逐渐减弱。忘记门是通过一个激活函数来实现，如式(12.17)所示：

$$f_t = \text{sigmoid}(W_f^T \times s_{t-1} + U_f^T \times x_t + b_f) \quad (12.17)$$

通过将上一隐藏层的输出信息 $s_{t-1}$ 与当前的输入 $x_t$ 进行线性组合后，利用激活函数，将函数值压缩，得到一个大小在0和1之间的阈值，当函数值越接近于1时，表示记忆体保留的信息就越多。当函数值接近于0时，表示记忆体丢弃的信息就越多。将上面的过程用图12.5来可视化表示。

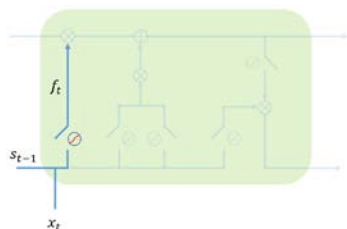


图12.5 忘记门逻辑设计

## 2. 输入门

输入门 (Input Gate Layer), 它决定了当前时刻的输入信息  $x_t$ , 有多少信息将被添加到记忆信息流中, 与忘记门的计算公式几乎一致, 输入门同样通过一个激活函数来实现, 如式(12.18)所示:

$$i_t = \text{sigmoid}(W_i^T \times s_{t-1} + U_i^T \times x_t + b_i) \quad (12.18)$$

## 3. 候选门

候选门 (Candidate Layer), 它用来计算当前的输入与过去的记忆所具有的信息总量, 其计算过程如式(12.19)所示:

$$C'_t = \tanh(W_c^T \times s_{t-1} + U_c^T \times x_t + b_c) \quad (12.19)$$

输入门与候选门的可视化效果图如图12.6所示。

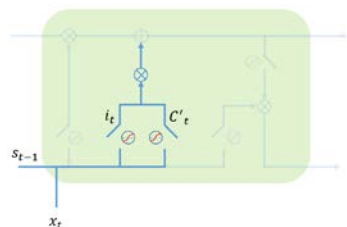


图12.6 输入门与候选门逻辑设计

记忆的更新由两部分构成: 第一部分是通過忘记门过滤过去的部分记忆, 大小为  $f_t \times C_{t-1}$ ; 第二部分是添加当前的新增数据信息, 添加的比例由输入门控制, 大小为  $i_t \times C'_t$ 。将这两个部分进行组合, 得到更新后的记忆信息  $C_t$  为:

$$C_t = f_t \times C_{t-1} + i_t \times C'_t \quad (12.20)$$

输入门与候选门的组合效果图如图12.7所示。



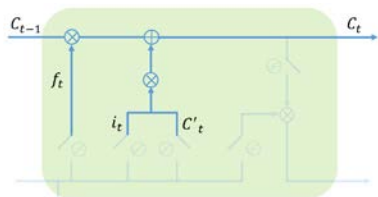


图12.7 输入门与候选门逻辑设计

4. 输出门

输出门（Output Gate Layer），它控制着有多少记忆信息将被用于下一阶段的更新中，即由 $C_t$ 求解 $s_t$ 的过程，输出门的计算公式满足：

$$o_t = \text{sigmoid}(W_o^T \times s_{t-1} + U_o^T \times x_t + b_o) \tag{12.21}$$

$o_t$ 是一个大小在0和1之间的权重值，传递给下一阶段的记忆信息为：

$$s_t = o_t \times \tanh(C_t) \tag{12.22}$$

输出门逻辑设计效果图如图12.8所示。

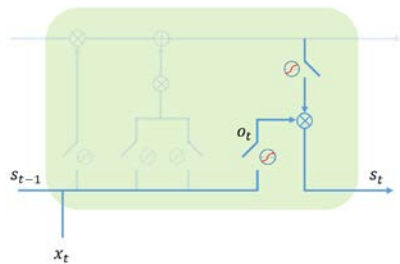


图12.8 输出门逻辑设计

我们把前面的知识点综合，得到如图12.9所示的全局效果图，图中展示了在第 $t$ 时间步，隐藏层中的数据流动和各“门”单元的交互影响。

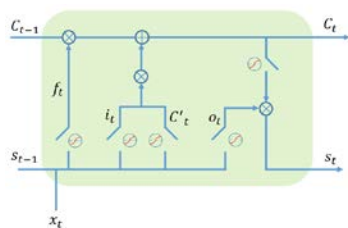


图12.9 LSTM隐藏层的逻辑设计图

上面分析了标准的LSTM网络的设计流程，但这并不是唯一的设计方式。事实上，

在实际的应用中，隐藏层的“门”单元设计，可能会由于要解决的实际问题不同，在设计上会有细微的差别，其中一种较流行的改进设计，是由Cho等人在2014年提出的门控递归单元网络（简称GRU）<sup>[6,7]</sup>，GRU在LSTM的基础上进行了简化，主要包括下面两个方面的改造：一是将输入门与忘记门组合为一个新的门单元，称为更新门（Update Gate）；另一处变动是将记忆单元 $C_t$ 与隐藏层单元 $s_t$ 组合为一个统一的单元，经过改动后的隐藏层效果图如图12.10所示。

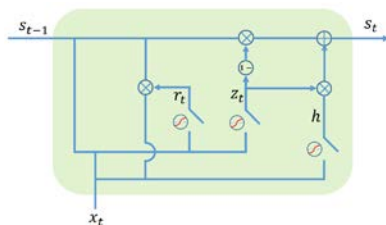


图12.10 GRU隐藏层的逻辑设计图

其中 $r_t$ 表示重置门（Reset Gate），它的值决定了过去的记忆与当前的输入的组合方式； $z_t$ 表示更新门，它控制着过去的记忆有多少能被保存，重置门单元 $r_t$ 、更新门单元 $z_t$ 和记忆单元 $s_t$ 的计算公式如下所示：

$$z_t = \text{sigmoid}(W_z^T \times s_{t-1} + U_z^T \times x_t + b_z) \quad (12.23)$$

$$r_t = \text{sigmoid}(W_r^T \times s_{t-1} + U_r^T \times x_t + b_r) \quad (12.24)$$

$$h = \tanh(W_h^T \times (s_{t-1} \circ r_t) + U_h^T \times x_t + b_h) \quad (12.25)$$

$$s_t = (1 - z_t) \circ h + z_t \circ s_{t-1} \quad (12.26)$$

## 12.4 结构递归神经网络

前面介绍了时间递归神经网络及其两个常用的变种网络LSTM和GRU，它是将数据通过在时间维度的展开，使得信息能够在时间维度上进行传递和积累，数据之间的联系表现为在时间维度上的前后依赖关系；结构递归神经网络则是将数据在空间维度上展开，并以一种树结构的形式展示，数据之间的联系表现为整体与局部之间的空间组合关系。

要将自然语言应用于机器学习算法中，首先需要将语言数字化，最简单的方式是One-Hot Representation，但这种方式具有维度高，且不能反映出单词间的内在联系

等缺点,为此,Hinton等人最早提出了Distributed Representation<sup>[8]</sup>,也就是为单词 $w$ 构建词向量表示。当前,词的向量表示算法已经非常成熟<sup>[9,10,11]</sup>,由词向量构成的向量集合构成了词向量空间,如图12.11所示。但语言的复杂性,不但体现在词与词之间的关系上,更体现在不同长度的单词,短语和句子之间的关系上,例如单词“consider”和短语“take into account”的含义是一致的,如果能够将任意长度的短语、句子都映射到同一个向量空间中,无疑将对语言的表达和理解起到非常关键的帮助作用。

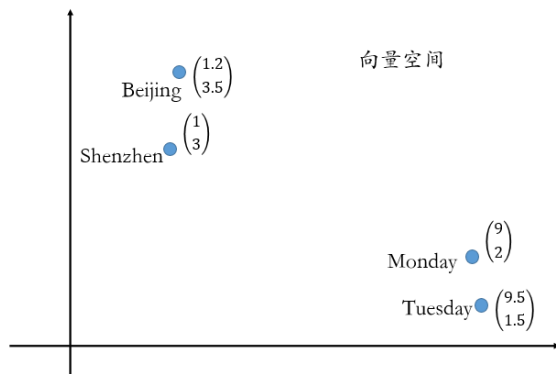


图12.11 词向量空间

Socher等人于2011年提出了结构递归神经网络<sup>[12]</sup>的模型,被广泛应用于句子的分块和语法处理中。在自然语言文本中,语法的规则是一种递归的结构,一个句子可以由名词短语和动词短语构成,而名词短语又可以继续分解为其他的短语子结构。以下的句子为例,考察如何由结构递归网络来构建句子的语法树。

句子:  $S = \text{the man saw a dog in the park}$

将句子按语法结构进行分解,得到如图12.12所示的语法树结构。

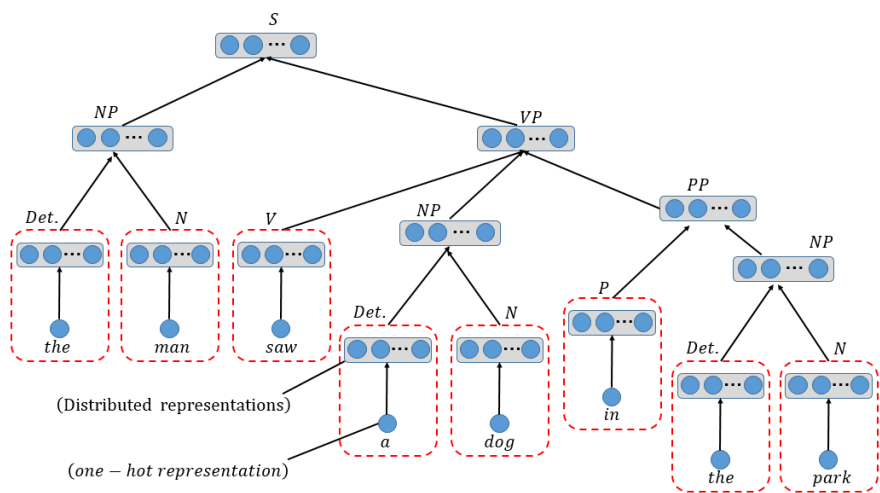


图12.12 句子的结构递归网络表示

整个句子的语法结构也可以用语法结构分解表来表示，如图12.13所示。读者可以对比这两种表示方式，从而加深对图12.12的理解。

符号	含义	短语
<i>S</i>	句子	<i>the man saw a dog in the park</i>
<i>NP</i>	名词短语	<i>the man</i>
<i>VP</i>	动词短语	<i>saw a dog in the park</i>
<i>Det.</i>	限定词	<i>the</i>
<i>N</i>	名词	<i>man</i>
<i>V</i>	动词	<i>saw</i>
<i>NP</i>	名词短语	<i>a dog</i>
<i>PP</i>	介词短语	<i>in the park</i>
<i>Det.</i>	限定词	<i>a</i>
<i>N</i>	名词	<i>dog</i>
<i>P</i>	介词	<i>in</i>
<i>Det.</i>	限定词	<i>the</i>
<i>N</i>	名词	<i>park</i>

图12.13 句子的语法结构分解表示

考察图12.12的语法树结构，为了得到整个句子*S*的向量表示，自底向上递归执行如下的操作：把小的短语表示合并为更大的结构表示，最终得到整个句子的向量。形式化来说，对于任意的输入数据 $c_1$ 和 $c_2$ ， $c_1$ 和 $c_2$ 既可以表示单词，也可以表示组合后的短语，通过合并两者的向量表示，得到组合结构 $p$ 的向量表示，把这个自底向上的

合并操作称为结构递归神经网络的前向操作（Feed-Forward Process），如图12.14所示。

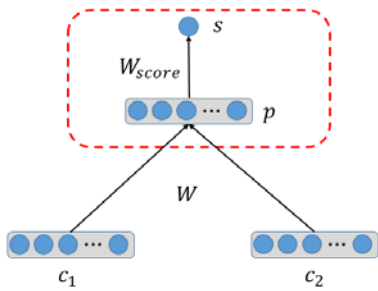


图12.14 对任意的输入数据，神经网络的前向操作将它们合并为更大的结构表示

由  $c_1$  和  $c_2$  得到  $p$  的向量表示的计算公式为：

$$p = \text{sigmoid} (W \times [c_1; c_2] + b) \tag{12.27}$$

$p$  与  $c_1$ 、 $c_2$  的向量表示都映射到同一个向量空间，经过式(12.27)的反复递归操作，最终得到整个句子  $S$  的向量表示，语法树中所有非叶节点的向量表示，与叶节点的词向量表示共用同一向量空间。

前面讲解了如何由词向量通过合并得到任意长度的句子向量表示。对于任意一个句子，一般预先并不知道语法树结构，词与词之间的组合方法可能有很多种，例如考察句子 “the cat is here”，图12.15展示了其中的两种组合方式。

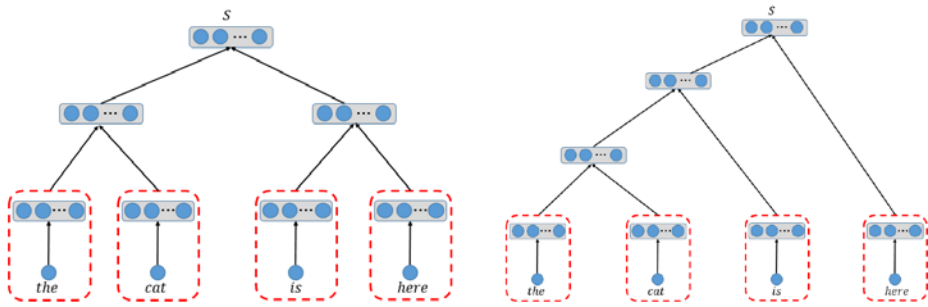


图12.15 对于同一个句子，可以通过不同的递归组合构建出不同的语法树

因此，一个亟待解决的问题是，需要选择一个评价标准，并按照这个标准，构建出句子的最优语法递归树。为此，在图12.14中，在进行合并操作后，还可以计算合并后的结构  $p$  的得分  $s$ ， $s$  的计算公式为：

$$s = W_{score} \times p \tag{12.28}$$

$s$ 的作用：衡量合并后的质量（ $s$ 越大则合并后的质量越高）；决定哪一对组合优先进行合并（取 $s$ 最大的组合方式进行合并）；计算合并后的句子的得分（总得分为所有合并结果的得分之和）。仍然以句子“*the cat is here*”为例，对应图12.15的两种组合方式的得分如下所示。

$$\text{左图总得分为：} score_L = s_{(1,2)} + s_{(3,4)} + s_{((1,2),(3,4))} \tag{12.29}$$

$$\text{右图总得分为：} score_R = s_{(1,2)} + s_{((1,2),3)} + s_{(((1,2),3),4)} \tag{12.30}$$

对应图12.15的两种组合方式的得分语法树如图12.16所示。其中，(a)图：首先将“the”和“cat”合并，得到 $p_1$ ，分数为 $s_{(1,2)}$ ，将“is”和“here”合并，得到 $p_2$ ，得分为 $s_{(3,4)}$ ，最后将 $p_1$ 和 $p_2$ 合并，得到整个句子，句子得分为 $s_{((1,2),(3,4))}$ ；(b)图将“the”和“cat”合并，得到 $p_1$ ，分数为 $s_{(1,2)}$ ，将 $p_1$ 和“is”合并，得到 $p_2$ ，得分为 $s_{((1,2),3)}$ ，最后将 $p_2$ 和“here”合并，得到整个句子，句子得分为 $s_{(((1,2),3),4)}$

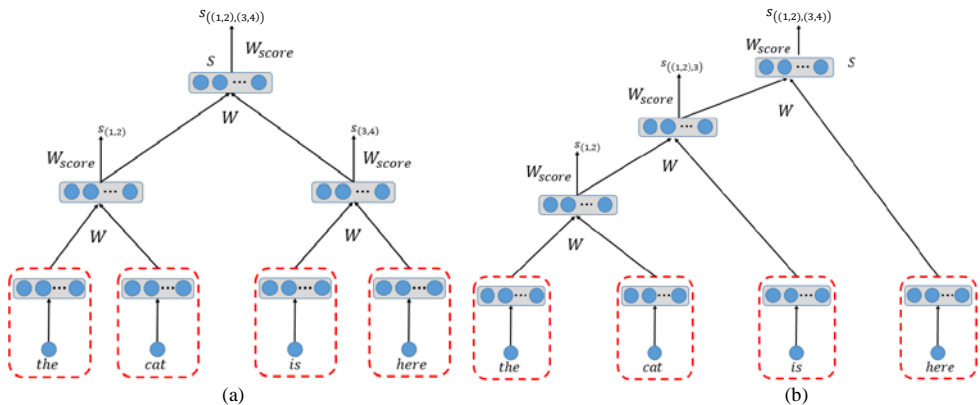


图12.16 对应图12.15的两种组合方式的得分语法树

采用beam search的方法构造出最优语法树，beam search的思想非常简单：对于当前的所有可能候选对，选择分数最高的候选对进行合并。通过一个具体的例子来考察，如何由结构递归网络和beam search构建一颗完整的语法树，如图12.17到图12.19所示。

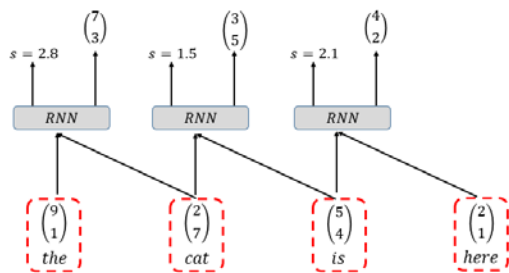


图12.17 第1步，相邻的两个单词合并，共有3个候选数据，在这3个候选数据中，“the”和“cat”的组合分数最大，因此，合并“the”和“cat”

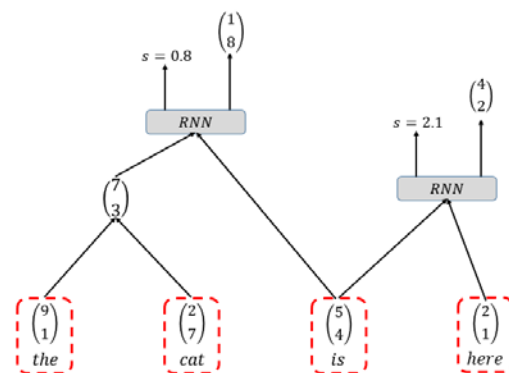


图12.18 第2步，由于“the”和“cat”合并为一个新的名词短语，当前只有两个候选对可供选择，其中，“is”和“here”的组合分数最大，将其合并进入最后一步操作

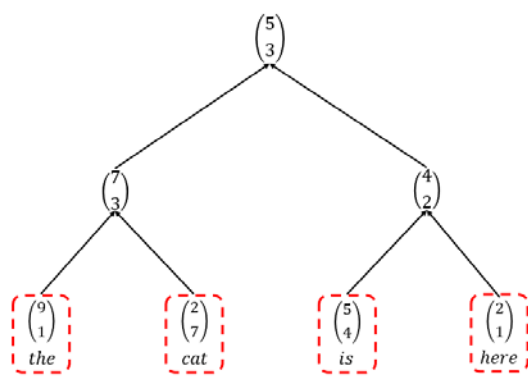


图12.19 第3步，当前只剩下唯一的候选对时，合并两个新的短语结构，最终得到句子的语法向量表示，语法树结构构造完成

## 12.5 应用：语言模型

语言模型是一个概率模型，利用该模型可以对任意给定的一个句子序列数据，得到该序列的合理性概率估计。在语音识别、机器翻译中扮演着非常重要的角色。语言模型的形式化定义为：设 $S$ 表示一个句子序列，它由单词序列 $\{w_1, w_2, w_3, \dots, w_n\}$ 构成， $n$ 表示句子的长度，则语言模型的任务是构建形如(12.31)式所示的概率模型：

$$p(S) = p(w_1, w_2, w_3, \dots, w_n) \quad (12.31)$$

### 12.5.1 N元统计模型

由条件概率公式，可以得到计算语言模型最简单的概率统计算法，如式(12.32)所示：

$$\begin{aligned} p(S) &= p(w_1, w_2, w_3, \dots, w_n) \\ &= p(w_1) \times p(w_2|w_1) \times p(w_3|w_1, w_2) \times \dots \times p(w_n|w_1, w_2, \dots, w_{n-1}) \end{aligned} \quad (12.32)$$

其中 $p(w_1)$ 表示单词 $w_1$ 出现的概率； $p(w_2|w_1)$ 表示在已知单词 $w_1$ 出现的条件下， $w_2$ 出现的概率；依此类推， $w_n$ 出现的概率取决于它前面的所有单词。可以想象，当 $n$ 比较大的时候，计算条件概率的复杂度是难以接受的。为此，俄罗斯数学家马尔科夫提出一种简化问题的假设：每一个单词仅与其前面的 $(n-1)$ 个单词相关，即满足：

$$p(w_t|w_1, w_2, \dots, w_{t-1}) = p(w_t|w_{t-(n-1)}, w_{t-(n-2)}, \dots, w_{t-1}) \quad (12.33)$$

把满足式(12.33)的假设称为 $(n-1)$ 阶马尔科夫假设，利用马尔科夫假设将式(12.32)化简为：

$$p(w_1, w_2, w_3, \dots, w_n) = \prod_{t=1}^n p(w_t|w_{t-(n-1)}, w_{t-(n-2)}, \dots, w_{t-1}) \quad (12.34)$$

把单词集 $w_{t-(n-1)}, w_{t-(n-2)}, \dots, w_{t-1}$ 称为 $w_t$ 的上下文环境，称式(12.34)对应的语言模型为N元统计模型。构建语言模型的关键是计算条件概率：

$$p(w_t|w_{t-(n-1)}, \dots, w_{t-1}) \quad (12.35)$$

考察如何有效求解上面的条件概率。给定训练的语料库文本，对任意的序列数据 $S$ ，要计算其概率 $p(S)$ ，需要通过在语料库中，计算前后单词组合的计数，也就是需要计算下面的统计量：



$$\text{count}(w_{t-(n-1)}, w_{t-(n-2)}, \dots, w_{t-1}, w_t) \quad (12.36)$$

$$\text{count}(w_{t-(n-1)}, w_{t-(n-2)}, \dots, w_{t-1}) \quad (12.37)$$

当训练的语料库数据量比较大时,根据最大似然估计的思想,式(12.35)的概率估计可以近似地化简为形如(12.38)式的计算形式:

$$p(w_t | w_{t-(n-1)}, w_{t-(n-2)}, \dots, w_{t-1}) = \frac{\text{count}(w_{t-(n-1)}, w_{t-(n-2)}, \dots, w_{t-1}, w_t)}{\text{count}(w_{t-(n-1)}, w_{t-(n-2)}, \dots, w_{t-1})} \quad (12.38)$$

N元统计模型为构建语言模型提供了一种有效的解决方案,但N元统计模型也存在着下面几个缺点:

第一,从计算的角度,N元统计模型的空间复杂度高。通常情况下,N元统计模型的空间复杂度是关于 $n$ 的指数函数,即 $O(|V|^n)$ ,其中 $|V|$ 表示语料库中词汇量的大小。在实际应用中, $n$ 的取值一般都比较小,事实上,当 $n=2$ 或 $n=3$ 时,模型已经有比较好的效果,随着 $n$ 的增长,资源消耗非常大,但效果却并没有得到显著提升。

第二,稀疏性。基于统计的语言模型需要知道模型中的所有参数,也就是所有词对共同出现的次数,但很多情况下,大部分词对并没有在语料库中出现,这样就造成了零概率问题;另一方面,当词对出现的次数很少时,数据结果的置信度也不高。针对稀疏性问题,研究学家已经提出了很多解决方法,其中平滑(smoothing)技术<sup>[14]</sup>,是比较有效并且应用最广泛的一种方法,下面以二元模型 $p(w_i | w_{i-1})$ 为例,介绍几种常用的平滑算法技巧。

### 1. 拉普拉斯平滑

拉普拉斯平滑(Laplace Smoothing),也被称为Add-one平滑,是最简单的一种平滑算法,它通过为每个词对出现的次数加1,保证每一个词对至少出现一次。因此,条件概率的求解公式为:

$$p(w_i | w_{i-1}) = \frac{1 + \text{count}(w_i, w_{i-1})}{\sum_{w_i} (1 + \text{count}(w_i, w_{i-1}))} = \frac{1 + \text{count}(w_i, w_{i-1})}{|V| + \sum_{w_i} (\text{count}(w_i, w_{i-1}))} \quad (12.39)$$

### 2. 古德-图灵平滑

古德-图灵平滑(Good-Turing Smoothing)通过折扣的方式来解决稀疏性问题。设在语料库中出现 $k$ 次的单词有 $c_k$ 个,语料库的文本大小为 $N$ ,那么有等式:

$$N = \sum_{k=0}^{\infty} k \times c_k \quad (12.40)$$

成立。这样，对于任意一个单词 $w_i$ ，若其出现的次数为 $k$ ，则其概率满足：

$$p(w_i) = \frac{k}{N} \quad (12.41)$$

前面提到，直接按频率来估计概率值，会存在零概率的情况，并且当出现次数比较小的时候，统计的结果不可靠。为此，Good和Turing提出通过折扣来下调出现频率比较低的单词的概率，首先引入 $d_k$ ， $d_k$ 的值为：

$$d_k = (k + 1) \times \frac{c_{k+1}}{c_k} \quad (12.42)$$

把单词出现的次数 $k$ 用 $d_k$ 来代替，容易验证 $\sum_k d_k \times c_k = N$ 成立，该等式表明用 $d_k$ 来替代 $k$ 并没有破坏数据的整体结构。由Zipf定律可知，一个单词出现的次数与它在频率表里的排名成反比，则 $c_{k+1} < c_k$ 成立，故在一般情况下，有 $d_k \leq k$ 成立，并且当 $k = 0$ 时，有 $d_0 = \frac{c_1}{c_0} > 0$ 成立。修改后的单词 $w_i$ 的概率计算公式为：

$$p_{gt}(w_i) = \frac{d_k}{N} \quad (12.43)$$

由于 $d_k \leq k$ ，因此修改后的概率比式(12.41)的概率值要小，起到下调置信度的作用。由于 $d_0 > 0$ ，因此即使对于没有出现的单词，其概率也会大于0，从而解决零概率问题。对于二元模型及更高阶模型，也采用相同的方式处理。

### 3. 线性插值平滑

古德-图灵平滑算法的一个缺点就是对于没有在语料库中出现的词对，它们的概率值是相等的。例如考察二元词对 $(w_i, w_j)$ 和 $(w_i, w_k)$ ，当满足 $\text{count}(w_i, w_j) = 0$ 且 $\text{count}(w_i, w_k) = 0$ 时，那么有等式：

$$p(w_j|w_i) = p(w_k|w_i) \quad (12.44)$$

成立，但在很多情况下，这个结果并不正确，例如，假设 $(best\ regards)$ 和 $(best\ happy)$ 在语料库中均没有出现，由式(12.44)，有：

$$p(regards|best) = p(happy|best)$$

成立，显然这个结果不符合常理，因为通常来说， $p(regards|best)$ 的值要远远大于 $p(happy|best)$ 的值。

为了解决这个问题,首先来考察在语料库中单词的统计规律性:考察一元组( $w_i$ )与二元组( $w_i, w_j$ )的统计关系,知道 $w_i$ 的出现次数必然比单词对( $w_i, w_j$ )的出现次数要多,也就是满足:

$$\text{count}(w_i) > \text{count}(w_i, w_j)$$

并且一元组( $w_i$ )的稀疏性问题远没有二元组( $w_i, w_j$ )严重;同理,考察二元组( $w_i, w_j$ )和三元组( $w_i, w_j, w_k$ )的关系,二元组( $w_i, w_j$ )出现的次数肯定比三元组( $w_i, w_j, w_k$ )出现的次数要多,即满足:

$$\text{count}(w_i, w_j) > \text{count}(w_i, w_j, w_k)$$

并且二元组( $w_i, w_j$ )的稀疏性问题同样没有三元组( $w_i, w_j, w_k$ )严重。

基于上面的规律,可以得到一个一般性的结论:低阶模型的零概率问题没有高阶模型严重,并且低阶模型的稀疏性要远好于高阶模型。基于这个结论, Jelinek和Mercer提出了线性插值的平滑算法(Jelinek-Mercer Smoothing)。线性插值平滑的基本思想是将高阶语言模型和低阶语言模型进行组合,从而达到平滑的目的。以二元模型为例,条件概率 $p(w_j|w_i)$ 的值可以通过下面的组合公式得到:

$$p(w_j|w_i) = \lambda_{(w_j|w_i)} \times p(w_j|w_i) + \lambda_{(w_i)} \times p(w_i) \quad (12.45)$$

#### 4. 卡茨平滑

卡茨平滑(Katz backoff Smoothing),也称为卡茨回退。它的基本思想是:如果高阶的语言模型概率为0,那么将回退到低阶,用低阶模型的概率来估算高阶模型,概率的估计呈现出层次结构性。下面以三元模型 $p(w_i|w_{i-2}w_{i-1})$ 为例:

(1) 当 $\text{count}(w_{i-2}, w_{i-1}, w_i) \geq T$ 时:直接使用式(12.38)的最大似然概率计算公式得到,即:

$$p(w_i|w_{i-2}w_{i-1}) = \frac{\text{count}(w_{i-2}, w_{i-1}, w_i)}{\text{count}(w_{i-2}, w_{i-1})} \quad (12.46)$$

(2) 当 $0 < \text{count}(w_{i-2}, w_{i-1}, w_i) < T$ 时:考虑到频率较小的词对存在置信度问题,结合古德-图灵平滑算法来估计,得:

$$p_{gt}(w_i|w_{i-2}w_{i-1}) = \frac{p_{gt}(w_{i-2}, w_{i-1}, w_i)}{p_{gt}(w_{i-2}, w_{i-1})} \quad (12.47)$$

其中 $p_{gt}(w_{i-2}, w_{i-1}, w_i)$ 和 $p_{gt}(w_{i-2}, w_{i-1})$ 的值仿照式(12.43)的古德-图灵平滑公式

得到。

(3) 当  $\text{count}(w_{i-2}, w_{i-1}, w_i) = 0$ ，则执行回退，使用  $p(w_i|w_{i-1})$  的值来估计三元模型，即：

$$p(w_i|w_{i-2}w_{i-1}) = \alpha \times p(w_i|w_{i-1}) \quad (12.48)$$

其中  $\alpha$  是一个归一化参数，由于条件概率满足：

$$\sum_{w_i} p(w_i|w_{i-2}w_{i-1}) = 1 \quad (12.49)$$

因此为了满足式(12.49)，需要保证：

$$p(w_i|w_{i-2}w_{i-1}) + p_{gt}(w_i|w_{i-2}w_{i-1}) + \alpha \times p(w_i|w_{i-1}) = 1$$

计算得：

$$\alpha = \frac{1 - p(w_i|w_{i-2}w_{i-1}) - p_{gt}(w_i|w_{i-2}w_{i-1})}{p(w_i|w_{i-1})} \quad (12.50)$$

这样把三元模型的概率估计问题回退到二元概率估计，对于二元概率估计问题，可以采用同样的回退策略，回退到一元模型。读者可以自行推导，这里不再详述。

N元统计模型还有很多巧妙的平滑技术，限于本书的篇幅，不在这里做进一步的探讨，更详细的算法细节读者可以参考相关文献[14,15,16]。

## 12.5.2 基于 LSTM 构建语言模型

上一节介绍了N元统计模型，通常情况下，当N值越大时，从理论上来说，模型的效果越好，但随着N的值越大，由于数据的稀疏性问题越来越严重，统计模型的效果却往往不理想。虽然采用平滑的策略能够起到一定的缓解作用，但没有很好解决这个问题。利用LSTM构建语言模型<sup>[17]</sup>，其基本思想是将句子拆分为单词的组合，句子中的每一个单词，按其出现的先后顺序作为输入层中每一时间步的输入数据，整个过程不需要关心输入数据是否在语料库中出现。考察下面两个句子。

$$S_1 = \text{the dog is eating} \quad S_2 = \text{the cat is eating}$$

当  $S_1$  在语料库中出现，但  $S_2$  没有在语料库中出现时，利用N元模型， $S_2$  的概率估计会出现零概率问题，它的结果往往与实际结果偏差较大。但仔细观察两个句子，我们发现， $S_1$  与  $S_2$  的意思实际上非常相近，它们的结果值应该满足  $p(S_1) \approx p(S_2)$ 。利用

LSTM来构建语言模型，我们的目的是学习词与词之间的前后依赖关系，而不是简单的统计规律，不但有效克服稀疏性造成的问题，也能够找出语言之间存在的联系。使用LSTM来构建语言模型，可以按照下面的步骤来执行。

### 1. 分词

几乎所有自然语言处理任务的第一个步骤都是对文本进行分词，对于英文文本，可以采用NLTK自带的`sent_tokenize`和`word_tokenize`来实现。对于中文文本，当前常用的分词工具包括jieba等分词包。分词完成后，将分词的结果，按照每一个单词出现的频率、顺序为每一个单词分配一个整数值。

### 2. 预处理

对分词后的结果进行预处理，常见的预处理包括：去除停用词，对没有在语料库出现的词和语料库中的非频繁词的处理。一般来说，为了提升效果，都把常用词汇限制在一个较小的范围内，比如20 000，对于非频繁词和没有出现的词，将其归类到一个Unknown\_Token中，在训练时作为相同的值来看待。其他预处理还包括是否限制训练数据的句子长度等。

### 3. 准备训练数据

将句子按图12.20的方式拆分为输入数据和输出数据。

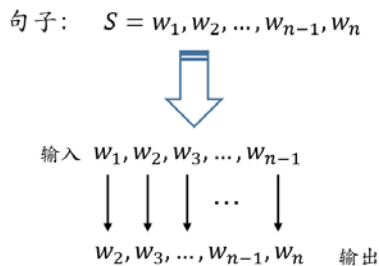


图12.20 将句子拆分为输入数据和输出数据

把文本通过索引映射，得到输入矩阵数据和输出矩阵数据，以图12.21为例的输入值向量为(5,103,32,5)，输出值向量为(103,32,5,2385)。

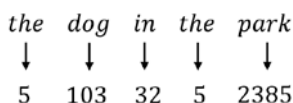


图12.21

## 4. 构建LSTM网络

参照12.3节构建LSTM模型，输入层以句子的输入数据作为输入，每一个单词对应一个时间步，在上面的例子中，时间总长度为4。在输入层与隐藏层之间添加Embedding层，将输入数据映射到低维空间中。隐藏层为了克服梯度消失的缺点，我们采用LSTM层构建，LSTM层的结构与原理请参考12.3节。输出层是一个softmax模型结构，通过输出层可以得到对应每一时间步 $t$ 的输出结果分布。完整的网络结构如图12.22所示。

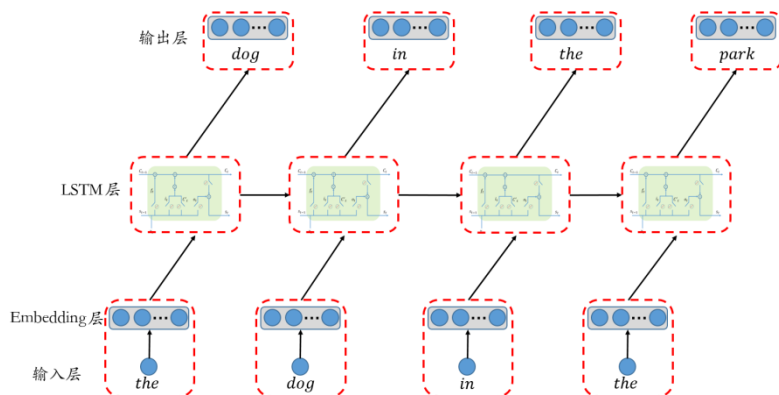


图12.22 利用LSTM构建语言模型的网络图

利用递归神经网络构建语言模型，最核心的设计是中间的隐藏层，图12.22设置了LSTM隐藏单元，通过前向操作，将句子中单词之间的联系信息，以记忆信息流的状态不断向后传递，影响每一个新输入数据的处理和每一阶段的输出，网络的前向传递操作可以通过scan函数来实现，代码如下所示。

```
ret, updates = theano.scan(
    fn = _step,
    sequences = [xx, mask, y],
    outputs_info = [T.alloc(0.0, n_samples, self.n_hidden).astype(theano.config.floatX),
                    T.alloc(0.0, n_samples, self.n_hidden).astype(theano.config.floatX),
                    None, None],
    non_sequences = [self.tparams['lstm_u'], self.tparams['lstm_w'],
                    self.tparams['lstm_b'], self.tparams['u'], self.tparams['b']],
)
```

在每一时间步 $t$ ，需要3个输入信息，分别是当前的输入数据 $x_t$ 、前一阶段的隐藏层传递信息 $s_{t\_prev}$ 和 $c_{t\_prev}$ ，有关 $s_t$ 和 $c_t$ 的概念请参考12.3节的详细讲解。处理完成后的输出数据同样由4部分构成，分别是 $s_t$ 、 $c_t$ 、 $o_t$ 和 $cost$ ，其中， $s_t$ 和 $c_t$ 分别表示记忆信息和候选信息，用来作为下一个时间步的输入， $o_t$ 表示当前时刻的输出值，它是一个向量值， $(o_t)_i$ 表示当前时刻 $t$ 取值为 $i$ 的概率，且满足：

$$\sum_{i=1}^n (o_t)_i = 1 \tag{12.51}$$

下面的代码展示了LSTM对数据信息的处理流程。

```
def _step(x_t, mask, y_t, c_t_prev, s_t_prev, lstm_u, lstm_w, lstm_b, u, b):
    prea = T.dot(x_t, lstm_u) + T.dot(s_t_prev, lstm_w) + lstm_b
    i = T.nnet.sigmoid(_slice(prea, 0, self.n_hidden))
    f = T.nnet.sigmoid(_slice(prea, 1, self.n_hidden))
    c_pi = T.tanh(_slice(prea, 2, self.n_hidden))
    o = T.nnet.sigmoid(_slice(prea, 3, self.n_hidden))

    c_t = f*c_t_prev + i*c_pi
    c_t = mask[:, None]*c_t + (1-mask)[:, None]*c_t_prev

    s_t = o*T.tanh(c_t)
    s_t = mask[:, None]*s_t + (1-mask)[:, None]*s_t_prev

    o_t = T.nnet.softmax(T.dot(s_t, u) + b)

    cost = -T.mean((T.log(o_t)[T.arange(y_t.shape[0]), y_t]) * mask)

    return c_t, s_t, o_t, cost
```

5. 数据并行与模型并行

数据并行：语言模型的训练数据是基于大规模的语料库来进行的，为了提升训练的效率，通常每一次的迭代训练都是按照批量更新的策略，取多条句子同时作为输入，但由于每一条句子长短不一，为此，我们首先需要将每一批次的所有句子规整为相同长度的向量，转化过程如图12.23所示：

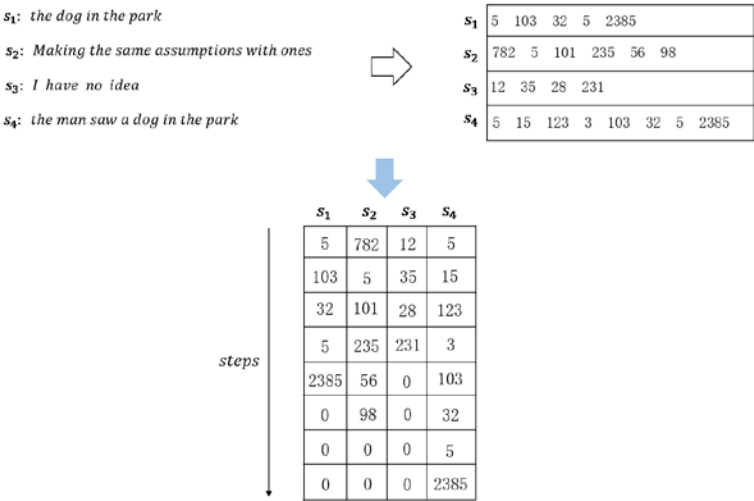


图12.23 将多条句子的词向量矩阵转化为等长序列的序列矩阵

模型并行：LSTM的核心是由式(12.17)、式(12.18)、式(12.19)和式(12.21)四个“门”运算构成，仔细观察这四个运算公式，我们不难发现，它们具有相同的预激活运算操作，并且预激活的系数矩阵具有相同的维度，因此，在执行计算的过程中，我们不需要单独进行四次运算操作，而是将四个运算同时并行执行，具体来说，我们可以构建新的系数矩阵 $\mathbf{W}$ 、 $\mathbf{U}$ 和 $\mathbf{b}$ ，满足：

$$\mathbf{W} = [W_f | W_i | W_c | W_o] \quad (12.52)$$

$$\mathbf{U} = [U_f | U_i | U_c | U_o] \quad (12.53)$$

$$\mathbf{b} = [b_f | b_i | b_c | b_o] \quad (12.54)$$

预激活操作使用公式 $\mathbf{W}^T \times s_{t-1} + \mathbf{U}^T \times x_t + \mathbf{b}$ 进行矩阵运算，计算结束后将同时得到四个“门”运算的预激活结果，通过并行化的矩阵运算，使得原来需要分四步执行的操作，只需一次运算就可以完成，计算过程如式(12.55)所示：

$$\mathbf{W}^T \times s_{t-1} + \mathbf{U}^T \times x_t + \mathbf{b} = (f_t \quad i_t \quad c_t \quad o_t) \quad (12.55)$$

## 6. 构建损失函数

语言模型采用交叉熵作为模型的损失函数，设当前的句子序列为：

$$S = w_1, w_2, \dots, w_{n-1}, w_n$$

则输入数据为 $x = (w_1, w_2, \dots, w_{n-1})$ ，输出 $y = (w_2, w_3, \dots, w_n)$ ，损失函数可表示为：

$$L(y, o) = -\frac{1}{n-1} \sum_{t=1}^{n-1} \ln(o_t)_{y_t} \quad (12.56)$$

其中 $y_t = w_{t+1}$ ，表示在第 $t$ 时刻的实际输出值。 $o_t$ 则表示在第 $t$ 时刻模型的预测输出向量， $(o_t)_{y_t}$ 表示在第 $t$ 时刻模型预测取值为 $y_t$ 的概率。

至此，利用LSTM构建语言模型的主要步骤已经简要讲解完毕，更详细的细节和实现，读者可以到本书配套的Github网站上查看本章的源代码。

## 参考文献：

- 
- [1] Bengio, Yoshua (2003). A neural probabilistic language model. Journal of Machine Learning Research 3: 1137–1155.
  - [2] D. Bahdanau, K. Cho, Y. Bengio. Neural Machine Translation by Jointly Learning to Align and



- Translate. 2014.
- [3] Alex Graves, Abdel-rahman Mohamed, Geoffrey Hinton. Speech Recognition with Deep Recurrent Neural Networks. 2013.
  - [4] Erik Bernhardsson. Recurrent Neural Networks for Collaborative Filtering. 2014.
  - [5] S Hochreiter, J Schmidhuber(1997). LONG SHORT-TERM MEMORY. Neural Computation 9(8):1735.
  - [6] J Chung, C Gulcehre, K Cho, Y Bengio(2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.
  - [7] Rafal Jozefowicz, Wojciech Zaremba, Ilya Sutskever. An empirical exploration of recurrent network architectures. Journal of Machine Learning Research. 2015.
  - [8] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by backpropagating errors. Nature, 323(6088):533–536, 1986.
  - [9] T Mikolov, I Sutskever, K Chen, GS Corrado, J Dean. Distributed representations of words and phrases and their compositionality. Advances in neural information processing systems, 2013. 3111-3119.
  - [10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. ICLR Workshop, 2013.
  - [11] Andriy Mnih and Geoffrey E Hinton. A scalable hierarchical distributed language model. Advances in neural information processing systems, 21:1081–1088, 2009.
  - [12] Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng, Christopher D. Manning. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. Proceedings of the 28th International Conference on Machine Learning (ICML-11). 2011. 129-136.
  - [13] R Socher, A Perelygin, JY Wu, J Chuang, CD Manning, AY Ng, CP Potts. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. EMNLP. 2013.
  - [14] Stanley F. Chen, Joshua Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. Computer Speech and Language (1999) 13, 359–394.
  - [15] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze: An Introduction to Information Retrieval, pages 237–240. Cambridge University Press, 2009.
  - [16] Buttcher, Clarke, and Cormack. Information Retrieval: Implementing and Evaluating Search Engines. pg. 289–291. MIT Press.
  - [17] T Mikolov, M Karafiát, L Burget, J Cernocký, S Khudanpur. Recurrent neural network based language model. interspeech. 2010.
  - [18] C Goller, A Kuchler. Learning task-dependent distributed representations by backpropagation through structure. Neural Networks, 1996.

## 卷积神经网络

卷积神经网络（Convolutional Neural Network，简称CNN），是深度学习技术中极具代表性的网络结构，它的应用非常广泛，尤其是在计算机视觉领域取得了很大的成功。它的诞生受到生物学的自然视觉认知机制的启发，1968年，诺贝尔生理学 and 医学奖得主D. H. Hubel和T. N. Wiesel，在研究猫脑皮层中用于局部敏感和方向选择的神经元时，发现其独特的网络结构，该结构可以有效地降低反馈神经网络的复杂性<sup>[1]</sup>。受此启发，1980年Kunihiko Fukushima提出了新识别机（neocognitron）的概念<sup>[2]</sup>，这也是第一个卷积神经网络的实现，自此之后，科学家们开始对CNN进行了深入研究和改进。1990年，LeCun等人发表论文<sup>[3]</sup>，确立了CNN的现代结构，后来又对其进行完善<sup>[4]</sup>。他们设计了一种多层的人工神经网络，也就是我们熟知的LeNet-5，该网络最初被用于手写识别分类。和其他神经网络一样，LeNet-5也使用反向传播算法进行训练<sup>[5]</sup>。

CNN相较于传统的图像处理算法的优点就在于，避免了对图像复杂的前期预处理过程（即大量的人工提取特征工作），也就是说CNN能够直接从原始像素出发，经过极少的预处理，就能够识别出视觉上面的规律。但受到当时环境的制约，比如缺乏大规模训练数据，以及计算机的计算能力跟不上，使得深度学习依赖的两大基础因素：“燃料”和“引擎”，都没有得到很好地满足。因此，LeNet-5对于复杂问题的处理结果并不理想，此后，随着神经网络进入了发展的低潮期，CNN的研究工作也一直没有突破性的进展。

从2006年起,在大数据和高性能计算平台的推动下,科学家们开始重新改造CNN,并设法克服难以训练深层CNN的困难。其中,最著名的是Krizhevsky等人在2012年提出的另一个经典的CNN结构<sup>[6]</sup>: AlexNet,该网络结构在图像识别任务上取得了重大突破,并以创纪录的成绩夺得当年的ImageNet冠军, AlexNet的整体框架与LeNet-5类似,但层数要更加深一些。

AlexNet取得成功后,研究人员又进一步提出了其他的完善方法,著名的CNN变种网络结构还包括:ZFNet<sup>[7]</sup>、VGGNet<sup>[8]</sup>、GoogleNet<sup>[9]</sup>和ResNet<sup>[10]</sup>4种。从结构看,CNN发展的一个显著特点就是层数变得越来越深,结构也变得越来越复杂,例如ILSVRC 2015的冠军ResNet,其深度是AlexNet的20多倍,是VGGNet的8倍多。通过增加深度,网络能够抽象出更深层、更抽象的特征。

但随着网络层数越来越深,结构越来越复杂,也使得网络变得难以优化和训练,很容易产生过拟合。因此,研究人员一方面通过增加深度来提取更多有效的特征,另一方面也提出了很多优化的方法来提升网络的效率和精度。本章将深入探讨CNN的设计原理,然后对CNN在最新的NLP领域上的应用进行介绍。

## 13.1 卷积运算

卷积操作是定义在两个连续实值函数上的数学操作,为了更好地理解卷积操作的意义,通过下面的一个例子来给出卷积的数学定义(本例子摘自参考文献[11])。

假设正在实时监控一首带有激光传感器的宇宙飞船,激光传感器在任意一个时刻 $t$ 都能产生一个实数输出 $x(t)$ , $x(t)$ 表示飞船在任意时刻 $t$ 的位置。一般来说,激光传感器的输出都带有噪音,为了减少噪音的影响,更准确地预测宇宙飞船的位置,可以对获取的数据进行加权平均。显然,对于相邻时间的输出结果,由于离当前时间较近,因此它们对结果的影响较大,相应的权重也应该较大;相反,离当前时间较远的输出结果,对当前时间的影响较小,权重值也应该相对较小,设权重函数为 $w(a)$ , $w$ 是一个窗口函数,它的值不会随着时间的改变而改变。加权平均后飞船的位置计算公式如式(13.1)所示:

$$s(t) = \int_{-\infty}^{\infty} w(a) \times x(t-a) da \quad (13.1)$$

这个操作被称为连续卷积操作,它定义了一个函数( $w$ )在另一个函数( $x$ )上的加权叠加,卷积操作通常也被记为:

$$s(t) = (w \times x)(t) \quad (13.2)$$

在式(13.2)中，通常把函数 $x$ 称为输入，函数 $w$ 称为滤波器（filter），或卷积核（kernel），得到的结果 $s$ 称为特征图，或者特征图谱（feature map）。

卷积操作满足交换律，即如果令 $p = t - a$ ，显然对于任意给定的 $t$ ，当 $a \rightarrow \infty$ 时，有 $p \rightarrow -\infty$ 成立；当 $a \rightarrow -\infty$ 时，有 $p \rightarrow \infty$ 成立，从而有：

$$\begin{aligned} (w \times x)(t) &= \int_{-\infty}^{\infty} w(a) \times x(t - a) da = - \int_{\infty}^{-\infty} w(t - p) \times x(p) dp \\ &= \int_{-\infty}^{\infty} x(p) \times w(t - p) dp = (x \times w)(t) \end{aligned} \quad (13.3)$$

因此，在其他一些参考书籍中，卷积操作可能会采用式(13.3)来表示，事实上两者是等价的。除了满足交换律外，卷积操作还满足分配律和结合律：

$$(w \times (x + y))(t) = (w \times x + w \times y)(t)$$

$$(w \times (x \times y))(t) = ((w \times x) \times y)(t)$$

图13.1展示两个方形脉冲的卷积运算操作，当滤波器 $w$ 从左侧一直滑动至右侧时，可以得到 $w$ 与 $x$ 的加权叠加结果 $s$ 。

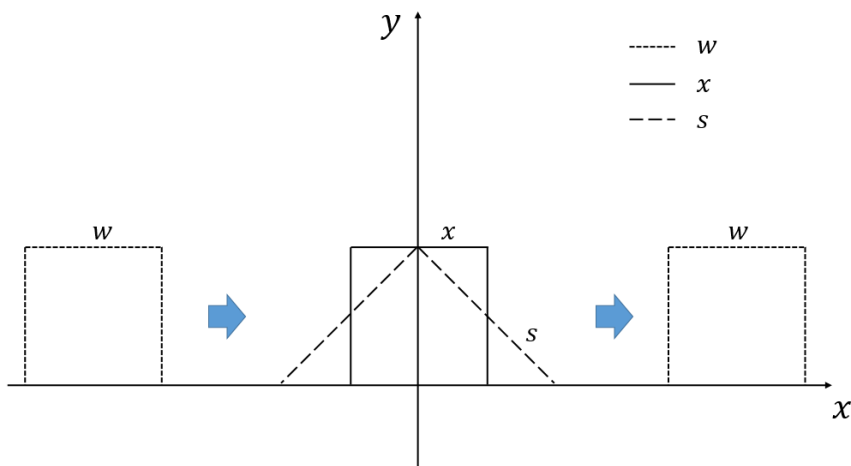


图13.1 两个方形脉冲的卷积运算

在前面的数学定义中，假设激光传感器产生的数据是一个连续的过程，则 $t$ 的取值可以是任意实数，但在计算机的处理场景下，连续卷积是不可能实现的，需要将数据离散化，事实上，一般情况下，也不需要记录任意时刻的数据信息，而是以一定的

时间间隔记录相邻的两个数据。假设有函数 $x$ 与函数 $w$ ，它们是定义在未知数 $t$ 上的函数，其中 $t$ 只能取整数，那么定义函数 $x$ 与函数 $w$ 的离散卷积公式为：

$$s(t) = (w \times x)(t) = \sum_{a=-\infty}^{\infty} w(a) \times x(t - a) \tag{13.4}$$

也可以把离散卷积的定义推广到高维空间上：

$$s(i, j) = (w \times x)(i, j) = \sum_m \sum_n w(m, n) \times x(i - m, j - n) \tag{13.5}$$

图13.2是在二维空间上应用卷积核的例子。

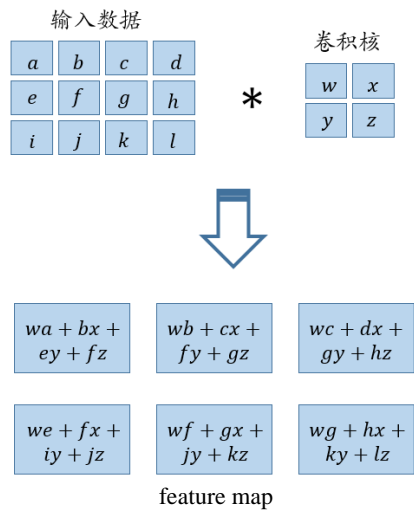


图13.2 二维数据上的卷积操作实例，卷积核是一个大小为 $2 \times 2$ 的二维结构，把卷积核应用到输入数据中，按照从左到右，从上到下的顺序分别执行卷积运算

离散卷积操作本质上是一个线性运算，因此卷积操作也被称为线性滤波。下面考察卷积操作在计算机视觉中的应用，给定一个大小为 $w \times h$ 的卷积核，将卷积核按照从左到右，从上到下的顺序扫描整个图像空间，对于大小为 $w \times h$ 的子图像空间，将子区域空间的像素值与卷积核执行卷积操作，获取该空间中相邻数据间的统计关系，通过这种操作，能够挖掘出图像的某些重要特征。

图13.3展示了利用卷积核提取图像的边缘，该卷积核也被称为高斯-拉普拉斯算子<sup>[12]</sup>。

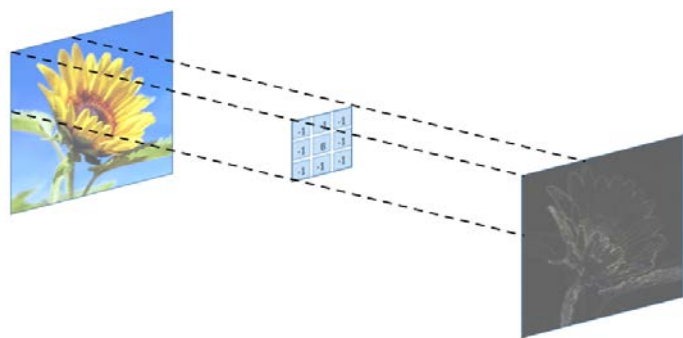


图13.3 利用卷积操作提取图像的边缘，卷积核的大小为 $3 \times 3$ ，该卷积核在图像处理中也是非常著名的滤波器，被称为高斯-拉普拉斯算子<sup>[12]</sup>

图13.4是采用均值滤波器对图像进行均值模糊处理后的效果。

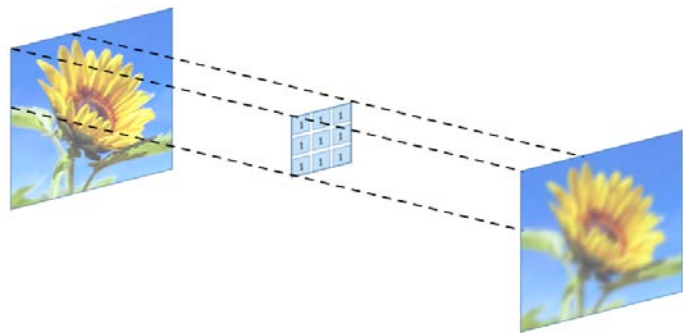


图13.4 在图像上应用均值滤波器得到的均值模糊效果，均值滤波器的每一个元素都为1

## 13.2 网络结构

在深入分析CNN的网络结构之前，先来考察CNN与前面章节的网络模型，特别是与全连通的前馈神经网络的异同点，首先是两者的相同点。

- 两者都是一个分层的深度网络结构，相邻网络层之间的神经元相互连接，而同一层的神经元之间，以及非相邻层的神经元之间没有关联，每一个神经元从上一层神经元中接收到输入数据，并进行线性加权（预激活），然后选择合适的非线性激活函数进行激活输出。
- 两者进行监督学习时，都是从输入端接收原始数据。在输出层定义损失函数，并通过最小化损失函数来调整权重，因此，前面各章提到的最优化方法和技巧，如反向传播、激活函数等同样适用于CNN网络。

那么CNN比传统的深层网络有哪些结构上的变化呢？传统的神经网络处理是一个全连接的网络，是在输入层接收原始数据，然后把数据传送到隐藏层，并不断向后抽象，如图13.5所示。

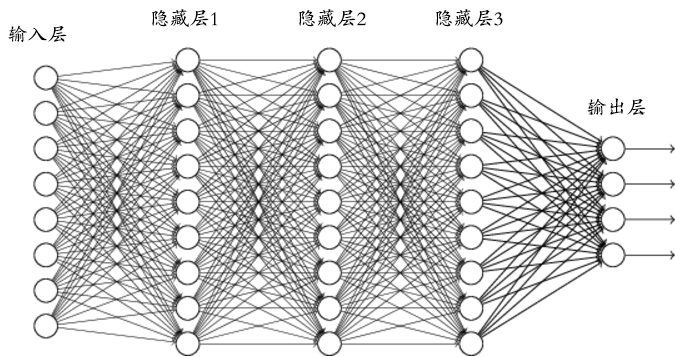


图13.5 全连接的前馈神经网络（图片摘自网络）

但全连接的深度网络存在很多缺点，如梯度消失、局部最优解、可扩展性差等。CNN的设计针对DNN的缺点做了相应的改进，并且避免了对输入数据复杂的预处理，可以直接对输入数据进行处理，实现了端到端的表示学习思想。图13.6是一个典型的CNN网络结构图，从图中我们可以看到，卷积网络在进入全连接层之前，已经经过了多个卷积层和池化层的处理。

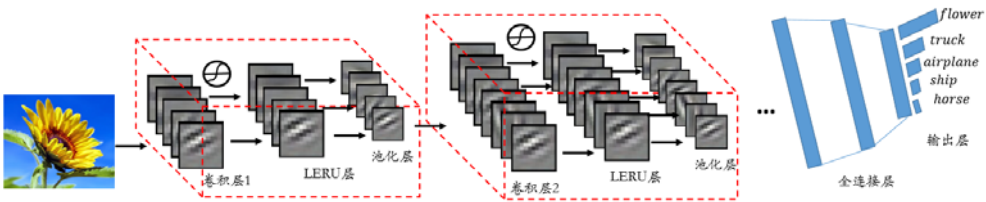


图13.6 VGGNet架构图，一种典型的CNN网络，主要由输入层、卷积层、LERU层、池化层、全连接层以及输出层等六部分构成

下面先简要概括CNN网络结构中较为重要的几个网络层设计，更详细的知识将在后面各节中介绍。

- **卷积层（Convolutional Layer）**：CNN网络的核心结构，通过局部感知和参数共享两个原理，实现了对高维输入数据的降维处理，并且能够自动提取原始数据的优秀特征。
- **激活层（Activation Layer）**：它的作用与传统DNN网络的激活层一样，把上

一层的线性输出，通过非线性的激活函数进行处理。第8章介绍的激活函数在这里仍然适用，既可以采用传统的sigmoid激活函数，也可以采用单侧抑制的ReLU激活函数。在深度学习领域，ReLU的单侧抑制激活函数普遍比传统的sigmoid激活函数效果要好。

- **池化层（Pooling Layer）**：也称为子采样层或下采样层，是CNN网络的另一个核心结构层。通过对输入数据的各个维度进行空间的采样，可以进一步降低数据规模，并且对输入数据具有局部线性转换的不变性，增强网络的泛化处理能力。
- **全连接层（Full Connected Layer）**：等价于传统的多层感知机MLP，经过前面的卷积层和池化层的反复处理后，一方面，输入数据的维度已经下降至可以直接采用前馈网络来处理；另一方面，全连接层的输入特征是经过反复提炼的结果，因此比直接采用原始数据作为输入所取得的效果更好。

## 13.3 卷积层

卷积层是CNN设计的核心，在具体讲解卷积层的结构和设计之前，首先来了解卷积层设计的3个动机。

**动机 1：减少参数数据的必要性。**

传统的前馈神经网络在数据处理上有一个致命的缺点，就是可扩展性非常差，例如将前馈神经网络应用于图像数据处理时，假设输入的是大小为 $1\,000 \times 1\,000 \times 3$ 的RGB图像，那么在输入层将由 $3 \times 10^6$ 个神经元构成。假设隐藏层同样由 $3 \times 10^6$ 个神经元构成时，对于全连接的前馈神经网络来说，输入层到隐藏层的权重参数约有 $3 \times 10^6 \times 3 \times 10^6 = 9 \times 10^{12}$ 个（不考虑偏移量参数）。可以想象，如果网络具有多个隐藏层，或者输入数据更复杂时，这个参数数据量还会进一步增加，显然，这个数量级的参数训练是一个极其耗时的过程。因此，如果能够有效减少神经网络的参数个数，对提升网络的训练效率是非常有帮助的。

**动机 2：生物学模拟的可行性。**

先来探讨大脑处理数据的机制。视野（Visual field）和感知野（Receptive field）是大脑处理外部世界，并与外部世界正确交互的关键。视野是指人的视觉系统可以看见的外部世界范围，但人的视觉系统在处理外部事物时并不是直接对全局进行处理。



Hubel和Wiesel的研究表明,视觉系统存在复杂的细胞分布,在处理外部世界的输入时,不同的细胞对输入的不同部分具有局部敏感性,比如某些细胞对运动数据敏感但对颜色不敏感,它能识别出整个视野的运动特征并忽略颜色的区别,这些局部敏感区域称为感知野或称为感受野。视觉系统通过将每一个局部感受野扫描整个视野区域,提取某一部分的特征信息,然后将不同的特征进行组合,达到全局处理的效果。生物学的这种局部处理方式卷积网络的卷积层设计提供了可行性依据。

### 动机 3: 提取深层特征的重要性。

CNN能够直接对输入数据进行预处理和特征提取,实现端到端的学习。那么卷积层是如何提取有效特征,从而提升网络的性能效果呢?长期以来,科学家和工程师都只能通过理论和数学的解析来强调卷积层的重要性。直到2014年,Zeiler和Fergus提出通过反卷积操作来可视化每一层提取的特征<sup>[13]</sup>,通过可视化,能形象地观察到CNN是如何从第一层简单的边缘特征提取开始,通过不断抽象和组合得到复杂的形状特征的过程,还能观察到通过调整卷积核参数,网络能提取到不同的特征信息,将多个不同的卷积核进行组合,能进一步加强网络的抽象能力。有关卷积网络的可视化知识,读者可以参考文献[13]。

卷积层的设计实际上由3个部分构成,经过卷积层的操作,不但能提取出有效的特征数据,还大幅度减少了参数的训练量。

#### 1. 局部连接

局部连接(Local Connectivity),也称为局部感知或稀疏连接。当处理高维度的输入数据,如图像、音频时,全连接的方式导致训练的时间和空间代价都非常高昂。此外,由大脑的局部感知野机制(动机2)可知,人对外界的认知是从局部到全局的过程,例如图像的空间联系也是局部的像素联系较为紧密,而距离较远的像素相关性则较弱。因而,每个神经元其实没有必要对全局进行感知,只需要对局部进行感知。具体来说,构造卷积层时,每一个神经元仅需要与上一层的部分神经元相连,这种方法也被称为**局部感知**或**局部连接**。以cifar10彩色图像作为输入数据,来考察局部连接的工作原理,如图13.7所示。

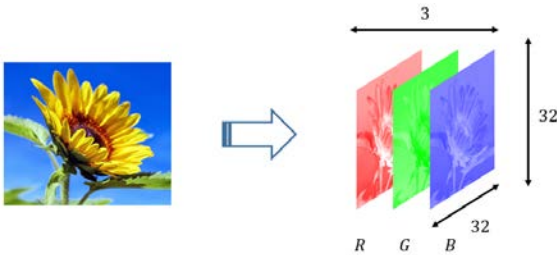


图13.7 RGB图像通过通道分离作为输入数据

每一幅 `cifar10` 图像是大小为  $32 \times 32 \times 3$  的 RGB 图像，局部感知野的大小对应于**卷积核**（`filter`，或称为滤波器 `kernel`）的大小，卷积核是一块子区域，如图13.8所示，定义了一个大小为  $5 \times 5 \times 3$  的卷积核（注意，卷积核的子区域是相对于宽度和高度来说的，卷积核的深度大小与原始数据要一致）。卷积层的每一个神经元只需要与卷积核对应的区域相连。

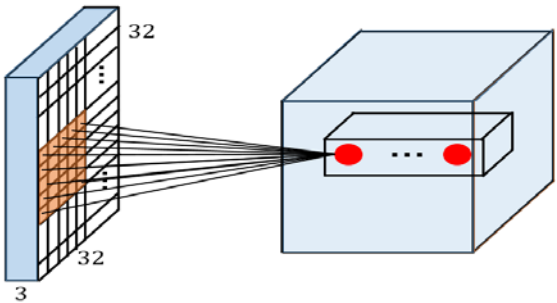


图13.8 卷积核作用于图像的过程

2. 空间位置排列（`spatial arrangement`）

前面讲解了局部连接的原理，下面具体考察卷积层的每一个神经元是如何与输入层进行交互的。事实上，卷积层的结构由四个超参数来决定，包括：卷积核的大小、卷积核的数量、步幅和补零。其中，卷积核的大小前面已经讲解，下面分别就另外3个超参数进行讲解。

（1）卷积核的数量：每一个卷积核能够提取某一部分的特征。显然，在绝大多数情况下，单卷积核提取的特征是不充分的，这时可以添加多个卷积核来提取多重特征，每一个卷积核与原始输入数据执行卷积操作后得到的神经元集合，我们称为特征图谱，如图13.9所示。每一个卷积核提取的特征都各有侧重点，因此，通常来说，最终的组合效果要比单卷积核的效果好很多。例如，在2012年的ImageNet竞赛中，Krizhevsky等

人夺得冠军的算法就用到了96个卷积核。

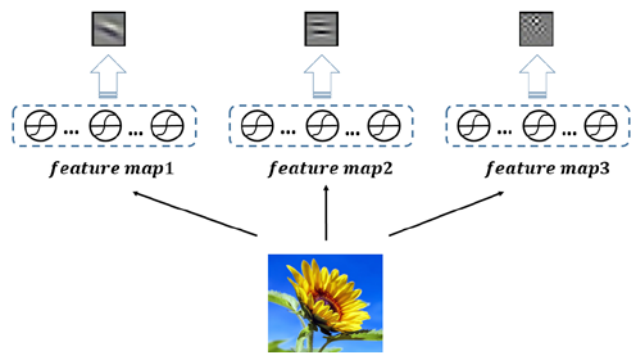


图13.9 对相同的输入数据，分别采用不同的卷积核进行操作，将提取到不同的特征数据

(2) 步幅 (stride): 需要指定卷积核在输入数据上的移动步幅，设步幅为大小 $s$ ，则表示卷积核在每一步会跳跃 $s$ 个像素。以一维数据为例，图13.10和图13.11分别展示了当步幅 $s$ 分别为1和2的时候对应卷积层神经元，假设卷积核大小为3，值为 $(1, 0, -1)$ 。

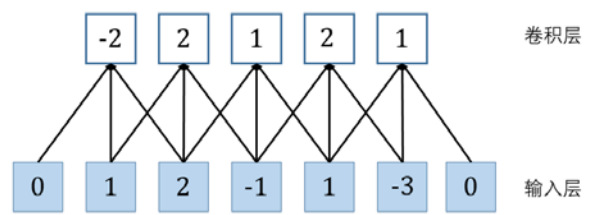


图13.10 当步幅 $s$ 分别为1的时候，卷积层的神经元空间分布

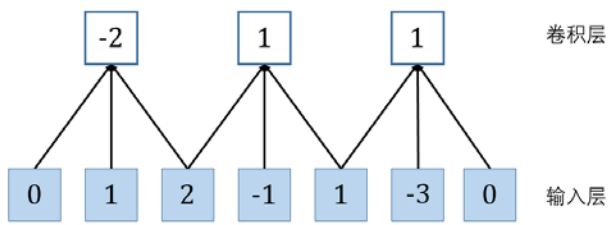


图13.11 当步幅 $s$ 分别为2的时候，卷积层的神经元空间分布

(3) 补零 (zero-padding): 通常被用来对边界进行处理。由于卷积核的大小不一定被输入数据的维度大小所整除，因此，可能会出现卷积核不能完全覆盖边界元素的情况。为了防止信息丢失，我们会采用补0的策略，使得在边界处的大小与卷积核吻合，例如，当步幅 $s$ 的大小为2时，若输入数据向量为 $(0, 1, 2, -1, 1, -3)$ ，卷积核为 $(1, 0, -1)$ ，卷积核并不能完全覆盖边界输入数据，这时候，一种解决方法就是在原始

的输入数据中添加额外的0元素，使得输入数据向量变为(0, 1, 2, -1, 1, -3, 0)。图13.12展示了这样的一个处理过程。

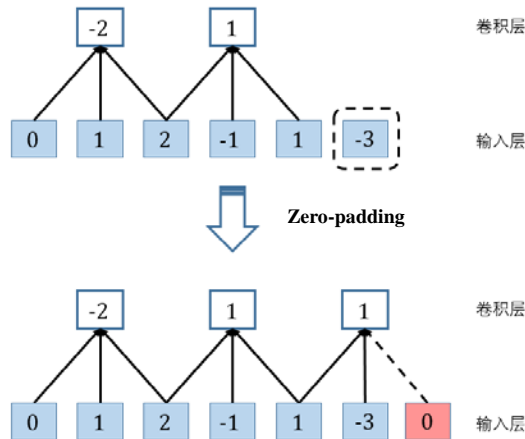


图13.12 通过在输入数据的边界处补0来覆盖全体的输入数据

综上所述，在构造卷积层时，对于给定的输入数据，当确定了卷积核大小、卷积核的个数、步幅和补零个数，那么卷积层的结构也能被完全确定。以一维数据为例，设输入数据的大小为 $w$ ，卷积核的大小为 $f$ ，步幅的大小为 $s$ ，补零的数目大小为 $p$ ，那么，对于每一个卷积核，由它与输入数据进行卷积运算后，得到对应的特征图谱所含有的神经元个数为：

$$\frac{(w - f + p)}{s} + 1 \tag{13.6}$$

对于高维数据，我们对每一维度分别按照式(13.6)进行计算。

### 3. 参数共享 (Parameter Sharing)

卷积层设计的第二个重要概念是参数共享，为什么需要参数共享呢？首先考虑这样一个问题，原始数据经过局部连接处理后参数的个数能降低多少呢？以 cifar10数据集的图像为例，每一个原始图像大小为 $32 \times 32 \times 3$ ，超参数设置包括：有100个卷积核、每一个卷积核的大小均为 $5 \times 5 \times 3$ 、步幅为1、没有补零。单独考察每一个卷积核，由式(13.6)，可知每一个卷积核对应的特征图谱含有的神经元个数为 $28 \times 28$ ，则对应的参数为 $28 \times 28 \times 5 \times 5 \times 3$ ，对所有的100个卷积核，总的权重参数为：

$$28 \times 28 \times 5 \times 5 \times 3 \times 100 = 5\,880\,000 \tag{13.7}$$

这个数据量虽然相比全连接的前馈神经网络有了一定的下降，但是数据量仍然很

大，依然没有办法满足高效训练的需求，而参数共享正是解决这个问题的有效方案。

参数共享是基于这样的一个假设：在数据挖掘中，数据在某一个位置 $(x1,y1)$ 的统计特性，同样适用于其他任意一个位置 $(x2,y2)$ 。也就是说，对于同一个卷积核，它提取到的特征应该适用于所有的位置，因此每一个卷积核对应生成的特征图谱神经元都将共享同一个参数列表。基于这一思想，我们把对应于同一个卷积核的所有神经元，用相同的参数来与输入层相连，如图13.13所示。

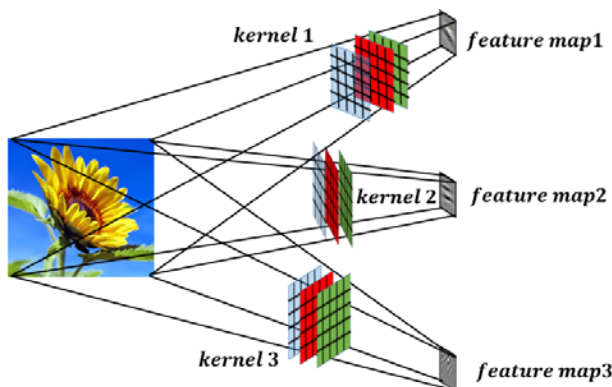


图13.13 利用参数共享策略，将卷积层神经元与输入数据相连，属于同一个feature map的神经元都将共用同一个权重参数矩阵

经过这样处理后，上面cifar10图像的参数数目为：

$$5 \times 5 \times 3 \times 100 = 7500 \quad (13.8)$$

注意，虽然参数共享大大降低了卷积层与输入层之间的权重参数，但并没有降低前向传播（forward propagation）的速度。到此为止，把标准的卷积层的设计剖析完毕。

## 13.4 池化层

池化层，也称为子采样层（Subsampling Layer），是CNN设计的另一个精髓，当通过卷积层提取物体的特征后，通常需要在两个相邻的卷积层之间插入池化层操作。

池化层函数是一个统计函数，以二维数据为例，如图13.14所示，输入数据的大小为 $W \times H$ 。给定一个池化滤波器，大小为 $w_1 \times h_1$ ，池化函数考察的是在输入数据中，大小为 $w_1 \times h_1$ 的子区域内，所有元素的某一种统计学特征。常见的统计特征，包括最

大值、均值、L2范数、加权平均等。

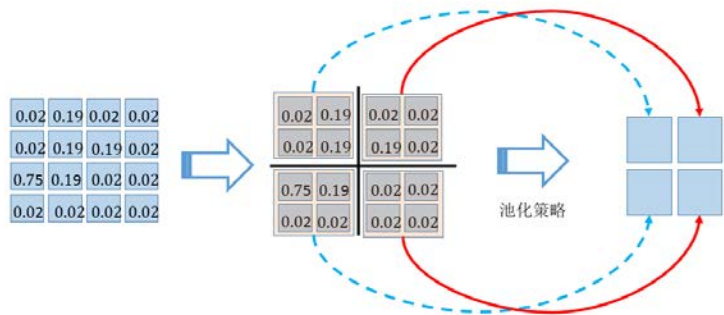


图13.14 池化操作，将池化滤波器内的所有元素用其统计量来替代

当前常用的池化策略主要包括最大池化函数和平均池化函数等。以一维向量数据(0.1, 0.2, 0.3, 0.2)为例，讲解两者在前向传播和反向传播时的运算过程。其中反向传播，也就是当采用BP算法来训练网络时，对池化层参数求导的操作。

(1) 最大池化函数 (max pooling)。

前向传播操作：取滤波器最大值作为输出结果， $\text{forward}(0.1, 0.2, 0.3, 0.2) = 0.3$ 。

后向操作操作：滤波器中的最大值元素不变，但滤波器中的其余元素为0，即满足 $\text{backward}(0.3) = (0, 0, 0.3, 0)$ 。

(2) 平均池化函数 (average pooling)。

前向传播操作：取滤波器平均值作为输出结果， $\text{forward}(0.1, 0.2, 0.3, 0.2) = 0.2$ 。

后向操作操作：滤波器中的所有元素都取平均值，即满足 $\text{backward}(0.2) = (0.2, 0.2, 0.2, 0.2)$ 。

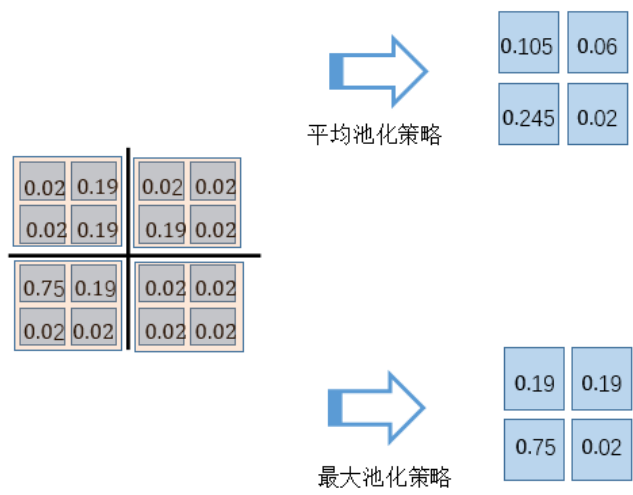


图13.15 两种池化策略得到的结果对比

池化层是CNN网络中非常重要的设计，池化层设计的作用主要有两个。

第一个，减少参数数量，防止过拟合。例如，当卷积层的输出大小是 $1\,000 \times 1\,000$ 时，如果池化滤波器的大小设置为 $2 \times 2$ 时，那么经过池化处理后的输出数据大小为 $500 \times 500$ 。也就是减少了4倍的数据量。

第二个，池化操作能保持局部线性变换的不变性<sup>[14]</sup>，也就是说，如果输入数据的局部进行了线性变换（平移、旋转）操作，那么经过池化操作后，输出结果并没有变化。

对于第二个作用，即线性变换的不变性，可以通过下面的例子来讲解，图13.16和图13.17分别是两个不同的输入数据经过相同的卷积层，非线性激活层和池化层后的结果，图中分别给出了两者的详细转换过程。

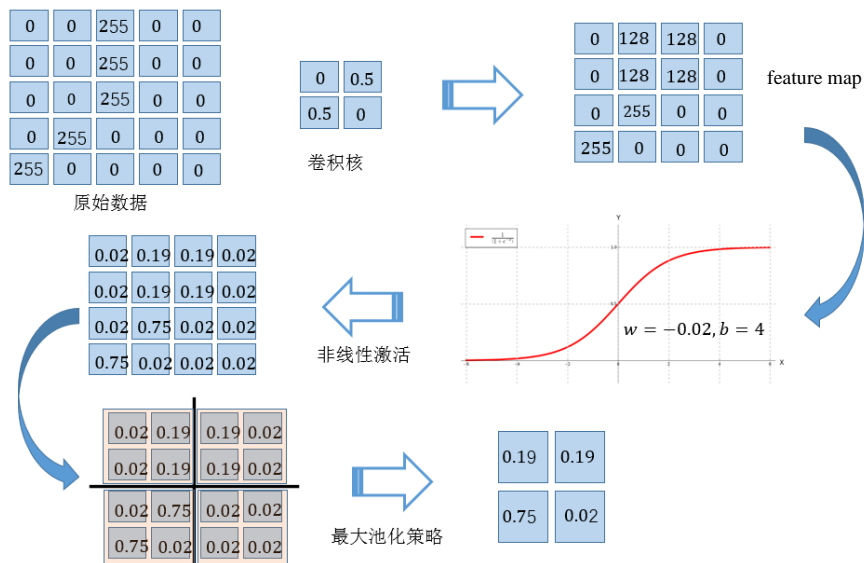


图13.16 输入数据首先与卷积核进行线性操作，得到特征图谱，然后经过非线性变换，激活函数，采用sigmoid函数其中的参数 $w = -0.02, b = 4$ 。激活函数对feature map的每一个元素执行非线性转换，输出结果将进入池化层操作，池化层采用 $2 \times 2$ 大小的滤波器，使用最大池化策略，最终得到池化结果输出，该结果将作为下一个卷积层或全连接层的输入数据

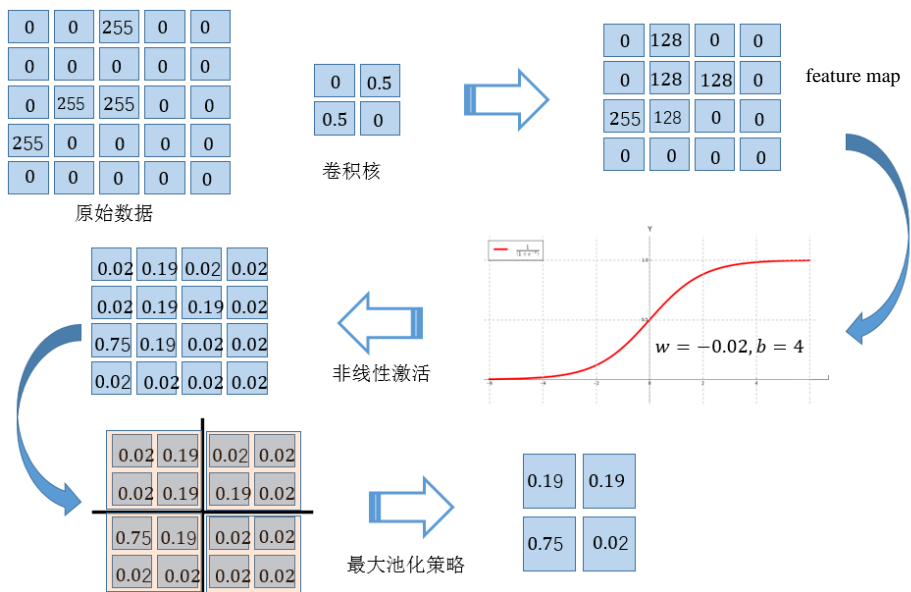


图13.17 对于另一个输入数据，采用与图13.16完全相同的策略，得到的结果输出与图13.16的结果完全一样



图13.18给出了详细的分析过程，局部线性变换的不变性在某些模式识别场景中特别有用，比如我们关心某一个物体是否存在，但并不关心它的具体位置。具体来说，比如在人脸识别的场景中，需要提取图像中的人脸，但并不需要准确知道人脸在图像中的位置，也就是说我们不关心人脸是出现在图像的左上角还是右下角，我们需要做的是能够检测到图像中是否具有人脸的特征信息，比如有一个眼睛在脸的左部，另一个眼睛在脸的右部等。

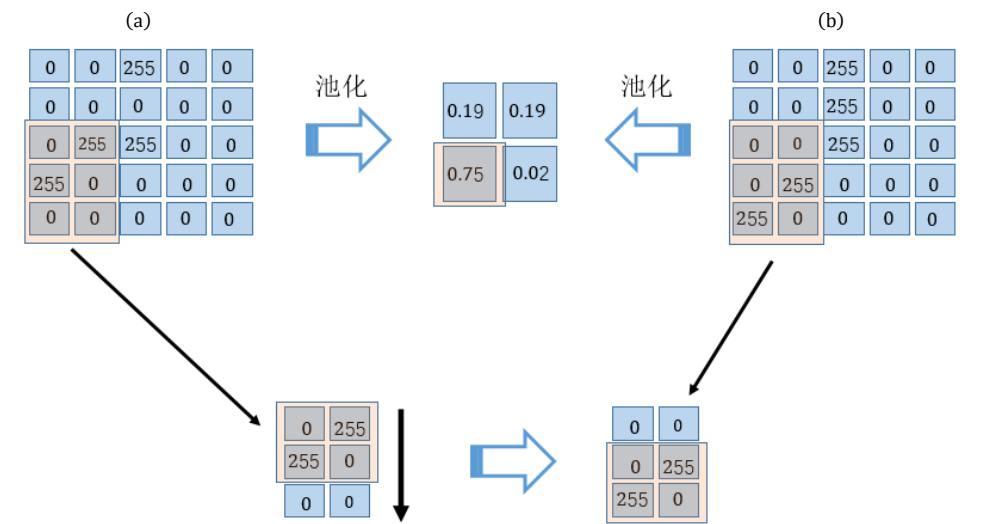


图13.18 仔细观察两个输入数据，我们发现两者之间最大的一个区别是左下角区域，输入数据(a)的左下角部分执行了向下平移的操作变成了数据(b)，而最大化池化策略，由于只考虑池化滤波器区域的最大统计量，线性变换没有破坏最大统计特征，因此它们都输出相同的结果0.75

### 13.5 应用：文本分类

前面分析了CNN网络在卷积层和池化层的设计原理，包括局部连接、参数共享和池化策略。但细心的读者可能会发现，前面都是以图像作为例子进行讲解，事实上，深度学习之所以能在近几年得到迅猛发展和普及，很大程度上得益于其在计算机视觉领域的突破，而CNN网络更是成为当前图像处理的标准处理算法。本节将把CNN网络应用到自然语言处理领域，并重点介绍它在文本分类中的应用。

从人类感知的角度出发，利用CNN处理文本并没有像用RNN（再次提醒，RNN是时间递归神经网络的简称）处理那么直观，因为对句子的理解，或者我们的阅读习惯，

都是从左向右的推进过程，而RNN特别适合于这种序列式的场景处理。把CNN应用于自然语言处理，主要是基于下面两个方面的考量。

第一，充分利用CNN优秀的特征提取能力。在图像处理中，利用CNN来提取特征是一种比较直观的操作，因为图像的局部特征可以应用于全局，但在NLP中，这种做法从直观上来说并不太合理。因为在一个句子中，这种局部的特征不一定适合于全局，从上一章我们知道，句子间单词的关系应该是一种由词性关系构成语法树结构。但令人惊喜的是，很多结果表明，将CNN应用于NLP取得的效果非常好<sup>[21]</sup>。

第二，运算效率的需要。CNN作为目前计算机视觉领域应用最广泛的模型，其核心操作是线性的卷积运算，很多线性代数运算库都实现了卷积的加速操作，将卷积加速的通常做法是将卷积操作转化为矩阵运算。以单通道输入、单卷积核为例，对于二维输入数据，传统的卷积运算是使用4个循环语句来完成，如图13.19所示：



图13.19 传统的卷积运算

将卷积转化为矩阵运算，需要将输入数据和卷积核都转化为矩阵的形式：对于输入数据，首先将输入数据按照局部感知野（即卷积核）的大小，按照从左到右，从上到下的顺序进行展开，每一行代表这样一个局部区域。对于卷积核，同样按从左到右，从上到下的顺序直接展开为一列，图13.20就是将上面的例子转化为矩阵运算的结果。

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.5 \\ 0.4 & 0.2 & 0.5 & 0.6 \\ 0.1 & 0.5 & 0.7 & 0.6 \\ 0.5 & 0.6 & 0.6 & 0.3 \end{pmatrix} \times \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} = \begin{pmatrix} 3.7 \\ 4.7 \\ 5.6 \\ 4.7 \end{pmatrix}$$

输入数据

卷积核

输出数据

图13.20 将图13.19的卷积运算转换为矩阵运算

对于多通道输入数据、多卷积核的卷积运算，同样可以采用这种转化方式，图13.21展示了一个三通道的二维输入数据和两个卷积核的原始卷积运算。

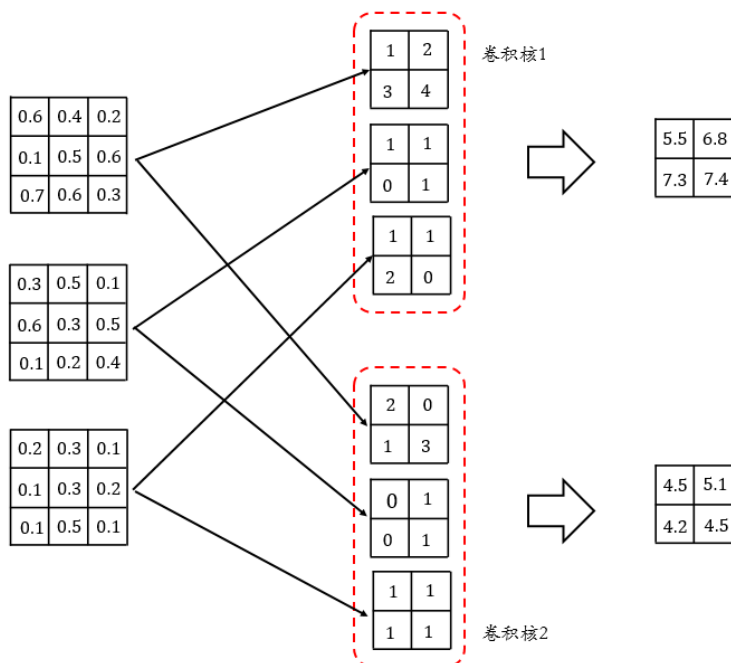


图13.21 更复杂的三通道、两个卷积核的卷积运算

将图13.21的卷积运算转化为如图13.22所示的矩阵操作。输入数据矩阵由三大矩阵块构成，每一个矩阵块对应于一个通道数据，卷积核矩阵的每一列对应于一个卷积核，读者可以将图13.22与图13.21进行比较，查看其变换过程。

$$\begin{pmatrix} 0.6 & 0.4 & 0.1 & 0.5 & 0.3 & 0.5 & 0.6 & 0.3 & 0.2 & 0.3 & 0.1 & 0.3 \\ 0.4 & 0.2 & 0.5 & 0.6 & 0.5 & 0.1 & 0.3 & 0.5 & 0.3 & 0.1 & 0.3 & 0.2 \\ 0.1 & 0.5 & 0.7 & 0.6 & 0.6 & 0.3 & 0.1 & 0.2 & 0.1 & 0.3 & 0.1 & 0.5 \\ 0.5 & 0.6 & 0.6 & 0.3 & 0.3 & 0.5 & 0.2 & 0.4 & 0.3 & 0.2 & 0.5 & 0.1 \end{pmatrix} \times \begin{pmatrix} 1 & 2 \\ 2 & 0 \\ 3 & 1 \\ 4 & 3 \\ 1 & 0 \\ 1 & 1 \\ 0 & 0 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 2 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 5.5 & 4.5 \\ 6.8 & 5.1 \\ 7.3 & 4.2 \\ 7.4 & 4.5 \end{pmatrix}$$

输入通道1      输入通道2      输入通道3      卷积核1    卷积核2      输出数据

图13.22 将图13.21的卷积运算转换为矩阵操作

将卷积操作转化为矩阵运算的最大优势是提高了计算的并行度，通过结合GPU等硬件设备来进行加速，可以使得运算效率得到显著提升，更多有关CNN网络的加速技巧可以参考文献[16,17]。

下面开始考察CNN是如何对文本进行分类的。

- 构造文本输入数据。

CNN在处理图像数据时，都是把图像的像素点作为输入数据，因此，要将CNN应用于自然语言处理中，首先要将文本转化为CNN可处理的数据形式，通过词向量，可以将每一个句子转化为矩阵的形式，如图13.23所示。

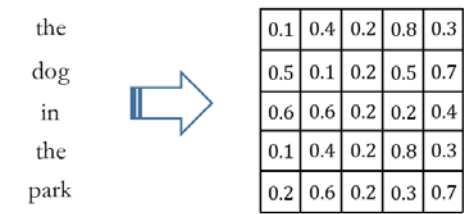


图13.23 将句子转化为矩阵

矩阵中的每一行代表一个单词的词向量表示，词向量可以使用word2vec、GloVe<sup>[18]</sup>，甚至是简单的one-hot向量，将句子的表示用式(13.9)来形式化定义，其中 $X_{1:n}$ 表示整个句子的组合由 $n$ 个单词构成， $X_i$ 表示第 $i$ 个单词的词向量：

$$X_{1:n} = X_1 \oplus X_2 \oplus \dots \oplus X_n \tag{13.9}$$

为了学习到文本之间的更多特征，可以采用多种词向量模型来表示同一个句子，即用多种不同的词向量算法来表示一个单词，相应得到了一个句子的多个不同的矩阵表示。参考彩色图像的处理方式，每一个矩阵等价于一个通道，类似于彩色图像的R、G、B通道，这样就把一个句子按通道进行了分离处理，如图13.24所示。

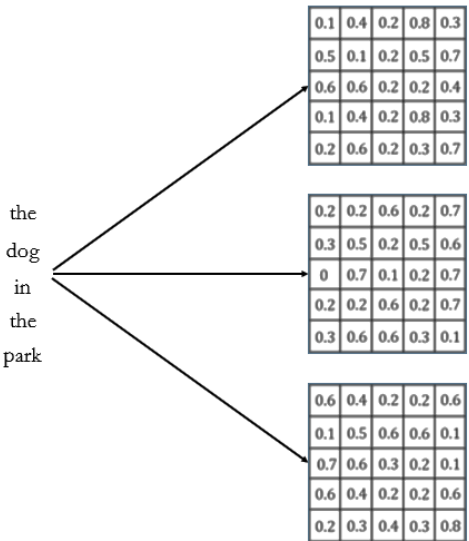


图13.24 采用不同的词向量来表示同一个句子，形成多通道的输入数据

• 卷积层和池化层的处理。

对于卷积层的设计，需要确定下面几个超参数，包括：卷积核的大小、卷积核的数量、步幅和补零。在图像处理中，卷积层是通过将卷积核放置在输入数据中，通过移动卷积核窗口来扫描整个区域，但在NLP中，这种做法并不合理，因为对于图像数据，每一个像素点是独立的关系，但由词向量构成的矩阵中，每一行代表一个完整的单词，因此，对文本数据构建卷积核时，卷积核的宽度应该刚好等于输入矩阵的宽度，卷积核在文本矩阵上的操作相当于一个滑动窗口，从上往下滑动，输入数据、卷积核和输出数据的大小关系仍然满足公式(13.6)，如图13.25所示。

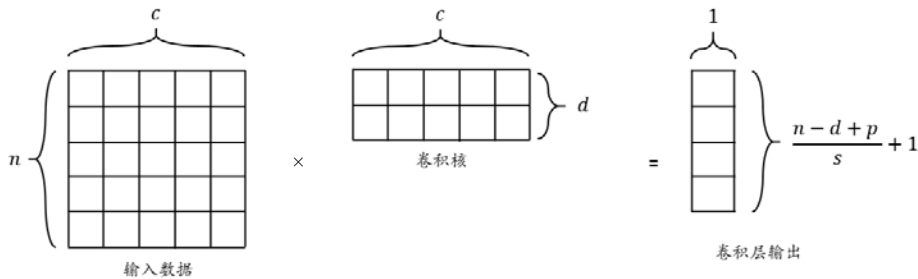


图13.25 输入数据、卷积核和输出数据的大小关系

对窗口大小为 $d \times c$ 的卷积核,当其作用于句子的第 $i$ 至第 $(i + d - 1)$ 个单词区间时,结果输出可以用式(13.10)来形式化表示:

$$C_i = f(W \times X_{i:i+d-1} + b) \tag{13.10}$$

其中 $W$ 是卷积核参数， $f$ 是激活函数，最终的卷积输出可以表示为（设步长为1，没有补零操作）：

$$C = (C_1, C_2, \dots, C_{n-d+1}) \tag{13.11}$$

可以采用多个大小不相同的卷积核来对同一个句子矩阵进行卷积运算。其原理与图像处理一样，通过不同的卷积核处理，可以提取不同的特征信息，有利于后面的特征组合。对池化层，可以将整个卷积的输出直接作用于池化函数。例如，将式(13.11)的结果直接应用于最大池化函数，得：

$$\tilde{C} = \max\{C\} \tag{13.12}$$

注意，式(13.12)将卷积层输出向量直接映射为一个数值，而不是像图像处理那样得到一个更小的矩阵，这样做的目的是希望能够捕获整个句子的最重要的全局特征值，而不是局部特征。文献[20]的实验结果指出，在文本处理中，大多数情况下，最大池化策略的效果要比平均池化策略的效果更好。

• 构建CNN训练网络。

将前面的知识点综合，可以得到图13.26所示的CNN文本分类模型。我们观察到，整个模型的结构非常简单，主要由3个部分构成。

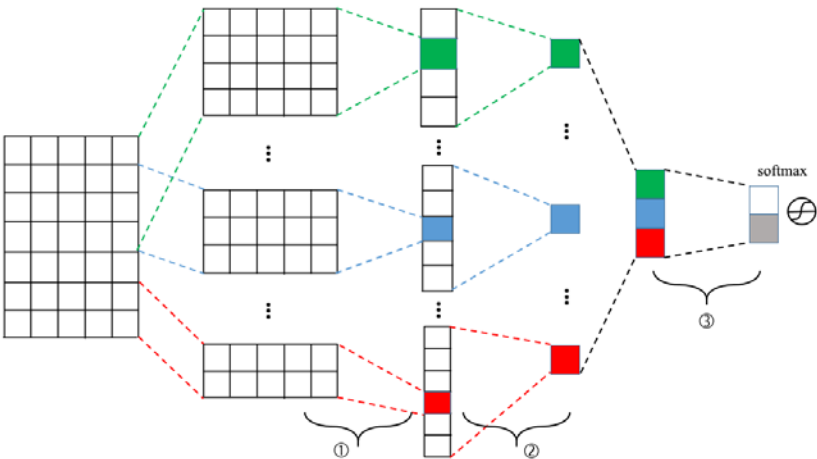


图13.26 简单的 CNN 文本分类模型

其一，卷积层和激活层：将卷积核作用于词向量矩阵，通过从上往下滑动扫描整

个输入数据,得到卷积输出,然后将激活函数作用于每一个卷积向量,得到式(13.11)的结果表示。

其二,最大池化层:运用最大池化策略得到每一个卷积输出向量的最大值,并将这些最大值合并,作为全连接层的输入特征向量。

其三,全连接层:图13.26展示的是最简单的单层感知机模型,只有输入层和输出层,通过softmax函数得到每一个文本的分类。

本节简要介绍了如何应用CNN网络对文本进行分类,上面的方法与步骤是基于文献[19]的讲解,也是最基本的模型思想。事实上,近年来,使用CNN对文本进行处理已经有了很多新的研究进展,其中,文本分类的研究还包括文献[21]、[22]、[23]等,它们在文献[19]的基础上,设计了更复杂的结构来捕获更深层的语义理解,包括对语义进行聚类、多通道的输入设计、更多的卷积层和池化层设计等。此外,文献[24]、[25]结合了LSTM和CNN两种网络模型来对文本进行分类。Denny Britz对深度学习在NLP领域的应用有一个比较全面的总结,更详细的细节,读者可以参考文献[26]。最后,读者也可以到本书配套的Github网站上查看本章的源代码。

## 参考文献:

---

- [1] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. The Journal of physiology. 1968.
- [2] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological cybernetics. 1980.
- [3] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a backpropagation network. Conference on Neural Information Processing Systems. 1990.
- [4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE. 1998.
- [5] R. Hecht-Nielsen. Theory of the backpropagation neural network. IJCNN. 1989.
- [6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, Imagenet classification with deep convolutional neural networks. Conference on Neural Information Processing Systems. 2012.
- [7] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. European Conference on Computer Vision. 2014.
- [8] K. Simonyan, A. Zisserman. Very deep convolutional networks for large-scale image recognition. International Conference on Learning Representations. 2015.

- [9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. CoRR, 2014.
- [10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385. 2015.
- [11] Ian Goodfellow, Yoshua Bengio and Aaron Courville. Deep Learning, Adaptive Computation and Machine Learning series. The MIT Press. 2016. ISBN-13: 978-0262035613.
- [12] Rafael C. Gonzalez, Richard E. Woods. Digital Image Processing, Pearson; 3 edition. ISBN-13: 978-0131687288.
- [13] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. ECCV 2014 pp 818–833.
- [14] Dominik Scherer, Andreas Muller, and Sven Behnke. Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. International Conference on Artificial Neural Networks. 2010.
- [15] Y-Lan Boureau, Francis Bach, Yann LeCun, Jean Ponce. Learning Mid-Level Features for Recognition. Conference on Computer Vision and Pattern Recognition. 2010.
- [16] Kumar Chellapilla, Sidd Puri, Patrice Simard. High Performance Convolutional Neural Networks for Document Processing. Guy Lorette. Tenth International Workshop on Frontiers in Handwriting Recognition, Oct 2006, La Baule (France), Suvisoft, 2006.
- [17] D Scherer, H Schulz, S Behnke. Accelerating large-scale convolutional neural networks with parallel graphics multiprocessors. International Conference on Artificial Intelligence. 2010.
- [18] J Pennington, R Socher, CD Manning. Glove: Global Vectors for Word Representation. EMNLP, 2014.
- [19] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuglu, P. Kuksa. 2011. Natural Language Processing (Almost) from Scratch. Journal of Machine Learning Research 12:2493–2537.
- [20] Y Zhang, B Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. arXiv preprint arXiv:1510.03820, 2015.
- [21] Kim, Y. Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, 2014. 1746–1751.
- [22] N Kalchbrenner, E Grefenstette, P Blunsom. A Convolutional Neural Network for Modelling Sentences. ACL, 2014.
- [23] Johnson, R., & Zhang, T.. Effective Use of Word Order for Text Categorization with Convolutional Neural Networks. arXiv preprint arXiv:1412.1058, 2014.
- [24] C Zhou, C Sun, Z Liu, F Lau. A C-LSTM neural network for text classification. arXiv preprint arXiv:1511.08630, 2015.
- [25] S Lai, L Xu, K Liu, J Zhao. Recurrent Convolutional Neural Networks for Text Classification. AAAI, 2015.
- [26] Denny Britz. understanding convolutional neural networks for nlp. 2015.